U.S Utility Application for Patent

METHODS AND TECHNIQUES FOR DELIVERING RICH JAVA APPLICATIONS OVER THIN-WIRE CONNECTIONS WITH HIGH PERFORMANCE AND SCALABILITY

Inventor: Coach K Wei and Zakir Magdum

Title of the Invention

Methods and Techniques for Delivering Rich Java Applications Over Thin-wire Connections With High Performance and Scalability

Field of the Invention

This invention relates generally to methods and systems for enabling World Wide Web-based or Internet-based computing, and in particular, relates to methods, techniques, devices and systems for delivering rich Java applications over thin-wire connections or channel, while providing high performance and extremely high scalability.

Identification of Related Patent Applications

The present utility application for U.S patent is related to the following U.S applications for patent, each filed on behalf of inventor Coach K Wei:

Serial Number	Title
60/256,594 (Provisional Application)	Methods and Techniques for Delivering Rich Java Applications Over Thin-Wire Connections With High Performance and Scalability
09/484,405	Computing Architecture

A copy of each applications is incorporated herein and attached hereto as Appendix; and in addition, the disclosure of each is incorporated in its entirely herein by reference. In particular, application 60/256,594 is the provisional application related to the present utility application.

Background of the Invention

Web-based or web-centric computing systems, architectures, devices and methods are disclosed in the above-identified patent applications filed on behalf of inventor Coach K. Wei.

The rapid growth of the web, including the Internet and the wireless web, brings tremendous challenges as well as opportunities. Though web-centric computing appears to be the inevitable future, there are great challenges not yet answered, which include the following:

- The capability to support compelling applications within unpredictable network environments;
- The overwhelming complexities of web application development and deployment;
- The rapid growth and divergence of client environments including web browsers, PDAs, Handheld computers, and the like

It is thus an object of the present invention to address and overcome these challenges.

Summary of the Invention

These and other objects of the invention are attained by the present invention. In particular, the assignee of the present invention, NexaWeb Technologies Inc., has developed a system and product, referred to herein as Nexel, that addresses these challenges. Nexel is an enabling platform that, for the first time, enables developers to write programs for one platform and deploy them anywhere -- to any device in any environment -- and empowers these programs with unmatched richness and performance even over thin network connection.

The following specification and drawings set forth detailed and enabling information regarding certain examples of the invention, which, in one embodiment, is referred to as Nexel Server for Java. Nexel Server (or, Nexel Presentation Server) is the core of the Nexel Platform. The functionalities of Nexel Server for Java are:

- To manage and execute Java applications on the server;
- To adapt the User Interface (UI) of these Java applications according to the capability of the client environment, such as screen size and client computing power.
- To deliver the UI of these Java applications to the client computer over network;
- To dispatch user events from the client device to corresponding applications.

Some characteristics and benefits of Nexel Server for Java are:

- Compatibility with existing applications: Nexel Server will be compatible with existing Java applications and will be able to run existing Java applications without code modification;
- Zero learning curve: the Nexel Platform will not change how developers write programs and will not require developers to learn anything new. Existing Enterprise Java Bean (EJB) components, application server environments, database and so on can be used without change.

- Extreme scalability: each Nexel Server will be able to run several thousands of Java applications concurrently.
- High performance: The Nexel Platform will deliver native-like performance over any network connection.
- Compatibility with Java 2 Enterprise Edition (J2EE) Application Server environments: Nexel Server will be able to run together with J2EE Application Servers in a plug&play fashion. See white paper "The New Economy Enterprise Computing Architecture" for how Nexel Server and J2EE Application Server are complementary to each other in the new economy enterprise computing.

To achieve these goals, Nexel Server design takes the following approaches, as disclosed herein:

- A thread-based computing model (or, a service model). Each event handler in an application is invoked as a separate thread that dies after the event handling is finished -- this is a demonstrably scalable solution;
- Some of the state information is maintained on the client side. This releases server from maintaining all the information in memory;
- Multiple Java Virtual Machines (JVM) are utilized for large scale computing. These JVMs can be on different machines. As a result, they form a server farm that can scale to any requirement.

Nexel Server Kernel conforms to the Java Servlet API Specification. It can run within a web container from any J2EE application server.

Further aspects of the invention are set forth in this specification and the drawings (which appear throughout the specification), which collectively present the invention at various levels of detail, from conceptual to specific examples of implementation.

Detailed Description of the Invention

In the discussion set forth hereinafter, for purposes of explanation, specific details are set forth in order to provide a thorough understanding of the invention. It will be appreciated by those skilled in the art that the present invention may be practiced without these specific details. In particular, those skilled in the art will appreciate that the methods described herein can be implemented in devices, systems and software other than those shown, and the examples set forth herein are provided by way of illustration rather than limitation. In other instances, conventional or otherwise well-known structures and devices are shown in block diagram form in order to facilitate description of the present invention.

The present invention also includes software steps that may be embodied in machine-executable software instructions, and thus the present invention includes a method that is carried out in a processing system as a result of a processor executing the instructions. In other embodiments, hardware elements may be employed in place of, or in combination with, software instructions to implement the present invention.

The specification and drawings are set forth in the following sections:

Overview
Architecture
Application Development and Deployment
Application Launching and Communication
Nexel Delivery Server - Platforms
Nexel Server Architecture
Nexel Core Classes
Nexel Java Foundation Classes
Technical Approach to Implement NJFC
Nexel Layout Manager
Nexel Network Engine
Additional Classes
Monitoring and Administration UI
Thread-Based Computing

The diagrams and textual descriptions set forth on the following pages explain the architecture, structure and details of certain embodiments of the invention. It will be appreciated by those skilled in the art that the present invention may be practiced without these specific details; that the methods described herein can be implemented in devices, systems and software other than those shown; that the examples set forth herein are provided by way of illustration rather than limitation; and that the scope of the invention is limited only by the appended claims.

2.1 Overview

The rapid growth of the web, including the Internet and the wireless web, brings tremendous challenges as well as opportunities. Though web-centric computing seems to be the inevitable future, there are great challenges not yet answered which include:

- The capability to support compelling applications within unpredictable network environments.
- The overwhelming complexities of web application development and deployment.
- The rapid growth and divergence of client environments including web browsers, PDAs, Handheld computers, and so on.

NexaWeb Technologies Inc's flagship product, Nexel, is an enabling platform that, for the first time, enables developers to write programs for one platform and deploy them anywhere -- to any device in any environment -- and empowers these programs with unmatched richness and performance even over thin network connection.

This document is a design specification for Nexel Server for Java. Nexel Server (or, Nexel Presentation Server) is the core of the Nexel Platform. The functionalities of Nexel Server for Java are:

- To manage and execute Java applications on the server;
- To adapt the UI of these Java applications according to the capability of the client environment, such as screen size and client computing power.

- To deliver the User Interface (UI) of these Java applications to the client computer over network;
- To dispatch user events from the client device to corresponding applications.

The engineering goals of Nexel Server for Java are:

- Compatibility with existing applications: Nexel Server will be compatible with existing Java applications and will be able to run existing Java applications without code modification;
- Zero learning curve: the Nexel Platform will not change how developers write programs and will not require developers to learn anything new. Existing Enterprise Java Bean (EJB) components, application server environments, database and so on can be used without change.
- Extreme scalability: each Nexel Server will be able to run several thousands of Java applications concurrently.
- High performance: The Nexel Platform will deliver native-like performance over any network connection.
- Compatibility with Java 2 Enterprise Edition (J2EE) Application Server environments: Nexel Server will be able to run together with J2EE Application Servers in a plug&play fashion. See white paper "The New Economy Enterprise Computing Architecture" for how Nexel Server and J2EE Application Server are complementary to each other in the new economy enterprise computing.

To achieve these goals, Nexel Server design takes the following approaches:

- A thread-based computing model (or, a service model). Each event handler in an application is invoked as a separate thread that dies after the event handling is finished. This is analogous to the model that Java Servlet is using. Java Servlet has been used in major web sites for handling millions of hits a day. It is a proven scalable solution;
- Some of the state information are maintained on the client side. This releases server from maintaining all the information in memory;
- Multiple Java Virtual Machines (JVM) are utilized for large scale computing. These JVMs can be on different machines. As a result, they form a server farm that can scale to any requirement.

Nexel Server Kernel conforms to the Java Servlet API Specification. It can run within a web container from any J2EE application server.

2.2 Architecture

Nexel brings an innovative approach to application delivery. Applications are developed using Java and standard J2EE platform. Once these applications are deployed with the Nexel Delivery Server, they can be used anywhere on the web. The NexWeb Delivery Server will present application UI on the client machine. The general architecture view is shown in Figure 1. Nexel consists of two parts. The server side part is called "Delivery Server." Its main responsibility is to interact with the application and extract its UI and

communicate with client part. The client side part is called "Client Kernel". It's main responsibility is to recreate a applications UI based on instructions from the server. Both these parts are described in detail in the following sections.

The Nexel server works with any web server that supports the servlet interface. It runs on top of the J2EE platform, which can be run on the Linux, Windows NT, Solaris, and HP-UX etc. operating systems. On the client side, we plan to develop players for playing these applications. These players will be implemented on the Windows platform with Internet Explorer and Netscape browsers. Also a proprietary player will be developed for Windows CE platform.

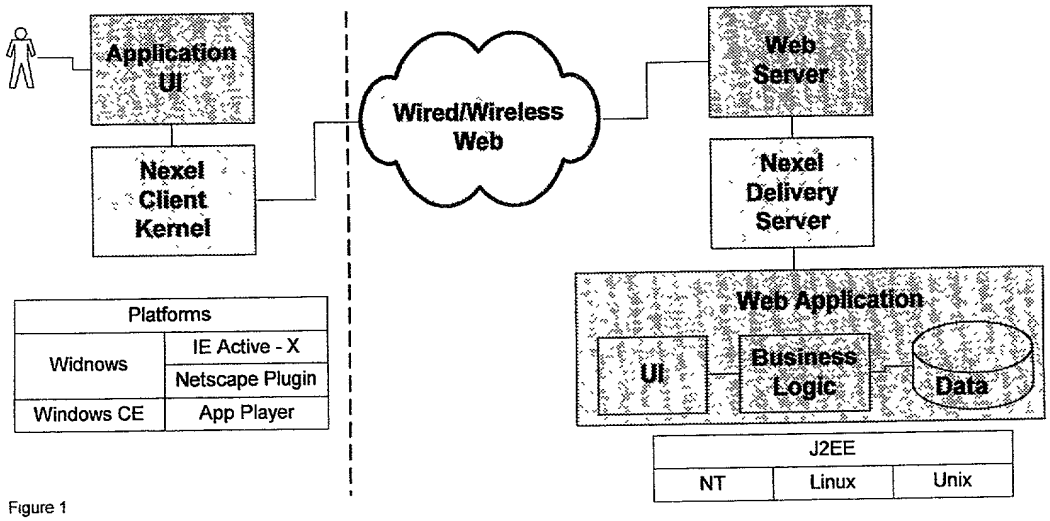


Figure 1

In addition to the delivery platform, NexaWeb also offers following tools:

Development Tools	Layout Manager
Deployment Tools	
Migration Tools	Code Analyzer

Each of these deliverable is described in detail in following sections. To understand the all these parts it is important to understand how applications will be developed and deployed.

2.2.1 Application Development and Deployment Process.

Nexel utilizes the existing process of developing applications for J2EE platform. In a three-tier architecture of an application, the data tier and business logic tier development process is unchanged. You can use any database or file system as data tier and you can develop the business logic using Enterprise Java Beans or any other Java technology or technology which can be called from Java. The UI layer has to be developed using Java Foundation Classes and Java. Many existing applications that use JFC for their UIs can be run in the Nexel environment without any modification. NexaWeb will produce migration tools for identifying problems in running existing code in the Nexel environment. The typical process for developing and deploying includes the following steps:

a. Develop the data layer using enterprise databases and file systems.
b. Develop business logic using enterprise java beans or using any other technology that can be called using Java.

c. Develop the application UI using Nexel's JFC. You can use Layout Manager to layout your screens at this point. Decide which events to process and connect the GUI to business logic. At this point you can use Nexel's validation objects to process events on the client side.
d. Install the Nexel application on the server using the Install Tool. You can customize the application UI for different client platforms using the Layout Manager. You can save these customizations into different files.

2.2.2 Application Launching and Communication.

It is also important to understand how applications are launched in Nexel environment. Communication between Delivery Server and Client Kernel go through the Web server using standard HTTP/HTTPS protocol. A socket-based communication should also be implemented for synchronous connection with the server. Application developer should be able to choose the approach at development time. The sequence of events that occurs when a user wants to launch application are

1. User open up the browser
2. User types in URL for the login HTML page. This page three items
   a. User Name
   b. Password for the user.

   Once the user enters the data, the form is submitted to the server. The server checks the user name and password and sends a page with available applications.
3. Each application has link setting with two query strings
   a. Application Name
   b. User Name

   The URL for each application will look like this http://websitename/UIServerServlet?Application=ApplicationName&User=UserName . Initially only the web servers that supports servlet-programming models will be supported. UIServerServlet will be installed on the web server and will provide functionality to pass the message from client to UI Server. Once the user clicks on an application URL it is passed on to the server.
4. Once the Nexel server receives commands to launch a application it will go through the following steps
   a. It will instantiate the application.
   b. Nexel server will also detect the type of client, which is invoking of the application and pass on to the application.
   c. A unique instance id will be assigned to application instance and pass on to the application.
   d. The application will run and it will talk to Nexel JFC API and create its UI. Nexel API will create a UI record format depending on the client type. At the end a record will be generated describing UI of the application.
   e. Nexel server encrypts the record if secure communication is used.
   f. Nexel server will respond to the user with the UI record.
5. Once the Client Kernel receives the UI record it will go through the following steps
   a. The client communication manager will read the UI record. It will decrypt the record if necessary.
   b. The client will read and parse the UI record.
   c. The client will convert the record into various UI commands and create the necessary components. Fill them with the data provided and setup event-monitoring mechanism.
   d. The application is now displayed on the client screen
6. The client Kernel will then monitor for all the events. When an event occurs it will find out whether it has to take any action. The action could be in two forms.
   a. Run the specified code on client side itself for that event. The code is specified using validation controls.
   b. Notify the server of the event.

The Client Kernel will take necessary action depending on the Application. If the event has to be notified to the server then the event processing happens on the server side. The server goes through the following steps once it receives the notification.

a. The server executes the method in the application, which is monitoring the events.
b. The server will monitor all the UI changes and converts them in UI record format.
c. It responds to the Client Kernel with that.

The client Kernel then goes through the process of updating UI. See steps 5.a through 5.d for details.

7. When user quit the application Client Kernel shuts down and notifies the server.
8. When the server receives message to quit it shut down the application instance.

When Synchronous communication is a requirement, the UI server will include a port number to which the client needs to establish the connection. Before creating any UI it connects to this port. Once the socket connection is established it is used for communicating later.

2.2.3 Nexel Delivery Server.

2.2.3.1 Platforms.

Nexel Server is implemented using Java technology. It should support all the major web servers on major platforms. Initially we will support Microsoft Internet Information Server (IIS) on Windows, Apache Web Server on Windows and Linux, Java Web Server on Windows and Linux.

Web Server	Initial Target Support	Phase 2 Target Support
Microsoft Internet Informatin Server	Windows NT/2000	
Netscape Enterprise Server		Solaris, Windows, Linux, HP-UX, AIX
Apache Web Server	Windows, Linux, Solaris	HP-UX, AIX
Java Web Server	Windows, Linux, Solaris	Solarix, AIX

2.2.3.2 Nexel Server Architecture

Nexel Server is composed of four sub systems, as illustrated in Figure 1. There core functionalities are shown in the following table. Note that two sub systems, Web Server Adapter and Java Servlet Engine, are standard and available freely. We do not need to Implement these two subsystems in Phase 1 development. Open source code, such as Tomcat/Jakarta project from Apache Software Foundation, can be used. We plan to modify these open source code by taking out the features that we don't need. A simplified Java Servlet.

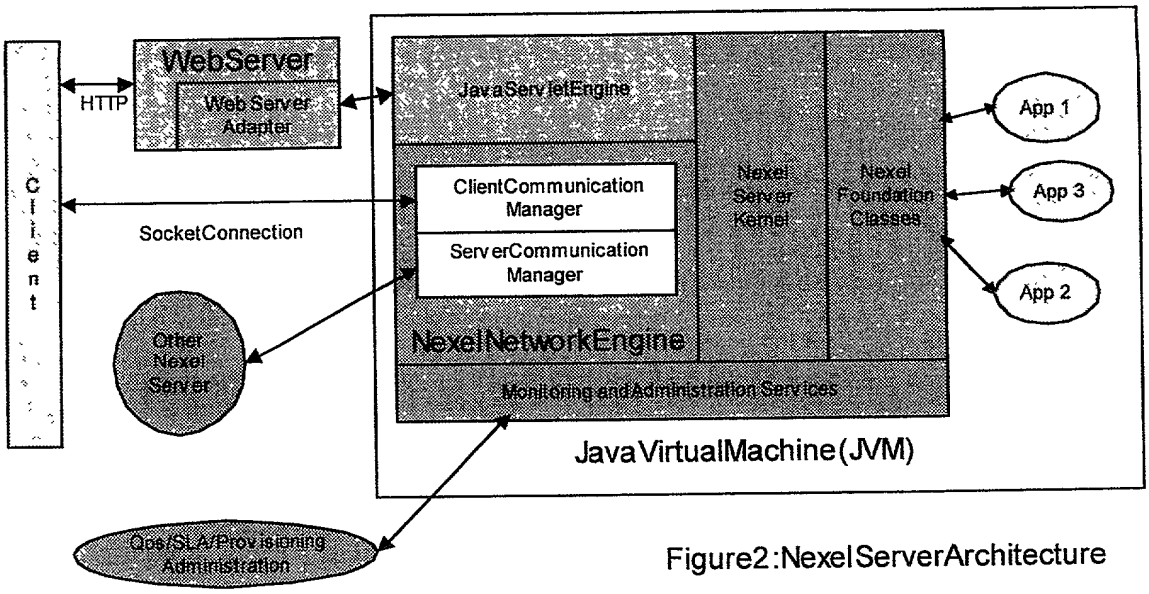


Figure2:NexelServerArchitecture

Name	Functionality	Note
Web Server Adapter	Communicates with the Java Servlet Engine, such as routing requests to and responses from the Servlet Engine.	Implementation is not needed for phase 1 development. We can use Java Web Server from the Tomcat Project.
Java Servlet Engine	Provides a standard interface for Nexel Server Kernel to interface with web servers. This standard interface is Java Servlet API. Provides thread management capability. Including: 1). Maintaining a thread pool; 2). Assigning or creating a thread to each request, and invokes the "service" method in Nexel Server Servlet from this thread.	The Servlet Engine can be any Servlet Engine, such as JRun and WebLogic. Implementation is not needed for phase 1 development. We can use Tomcat/JWSDK.
Nexel Server Kernel	Instantiates other Nexel components such as AppManager, ConnectionManager, EventManager and so on. Keeps these components within memory all the time; Launches applications; If necessary, launches another Nexel Server. Event (comes from Servlet requests) dispatching. Generates and sends application user interface description into Servlet responses. If necessary, re-do application user interface layout according the client display characteristics.	Written entirely in Java. Need to be implemented.
Nexel Network Engine	Accepts and creates synchronous socket connections to clients directly, by passing the Servlet Engine and Web Server; Thread management: Maintain a separate thread pool. Whenever a new request is received, creates a ServletRequest and a ServletResponse object, calls the servlet "service" method within a separate thread; returns this thread back to thread pool after the	Written entirely in Java. Needs to be implemented. Open source code from Tomcat project will be helpful.

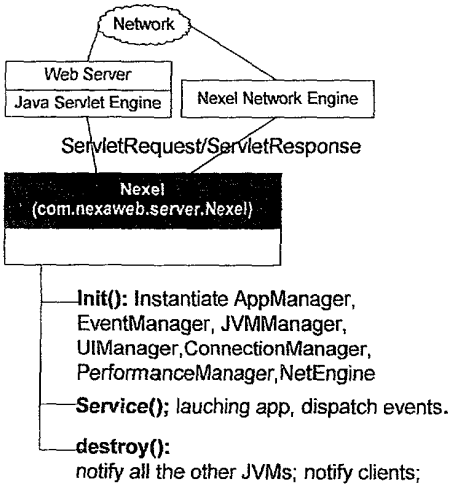
	"service" method returns; Communicates with other Nexel Server instances within other Java Virtual Machines through socket connections. The goal of such communication is to route requests/responses to different JVMs such that one JVM won't be overloaded.	
Monitoring and Administration Service	The Monitoring and Administration Service is responsible for keeping track of information about an Application Instance. Following information needs to be tracked. a. Application Name. b. Application Instance Identifier. c. User accessing the application. d. Client machine information. I. Machine IP II. Device Type. III. Connection Speed. e. Application start time. f. Application last access time. g. Memory usage. h. Active Screen Name	

Engine, a simple Java Web Server, and adapters from IIS, Apache, Netscape Web Server, will be shipped as part of our product.

Nexel Server Kernel is composed of three major parts:

2.2.3.2.1 Nexel Core Classes

Nexel(com.nexaweb.server.Nexel), AppManager,EventManager,JVMManager,UIManager, ConnectionManager,PerformanceManager.
Class com.nexaweb.server.Nexel: This class is the main class. This class extends javax.servlet.GenericServlet. As a result, it should run with any Java Servlet Engine. Figure 3 gives an overview of the functionalities of this class.




```
Nexel.Service(ServletRequest, ServletResponse):
 store SevletResponse to ConnectionManager;
 inspect request:
          if(JVM!=thisJVM)
                  pass request to NetEngine;
        else {
          if(appname!=null) {
            if(PerformaneManager.isOverLoaded())
                  pass request to NetEngine;
           else launchApp();
           }
          else dispatchEvent().
          }
 remove SevletResponse from ConnectionManager;
```

Figure 3: Nexel Servlet Overview

Class com.nexaweb.server.AppManager: This class manages all the applications within this Java Virtual Machine. Each application is represented as an instance of Class com.nexaweb.server.Application.

AppManager maintains a table of applications that can be searched by application ID. It also provides methods for getting the current application.
Class com.nexaweb.server.Application: This class maintains the persistent data between different threads within one application. It maintains the application ID, a Hashtable of components(Objects) that each component can be located by a unique component ID, a Hashtable of event listeners that each listener can be located by a component ID and an event ID. EventManager and other objects will use these IDs to get access to their interested objects.
The Application class should also maintain an event queue for processing events coming from different threads. See 4.3 Thread-based computing for further information.
Class com.nexaweb.server.EventManager: this class maintains a map between event ID and the actual event type. It also does the actual event dispatching: for a ServletRequest, it retrieves appid (application ID). Then it finds the application instance from AppManager by using the appid. It also retrieves the event ID and component ID from the ServletRequest. As last, it constructs corresponding Java Abstract Window Toolkit (AWT) or Swing events and sends the event to the Application object for processing.
Until the Application object finished the event processing, EventManager should make the current thread idle instead of returning to its caller. After the Application object finished processing this event, the current thread will be destroyed (or recycled).
Class com.nexaweb.server.JVMManager: this class maintains a Hashtable of running Java Virtual Machines (each of which has Nexel Server running). The key to the Hashtable is the IP address of the host where the JVM is running and the main port number that the Nexel Network Engine is assigned to listen to during its startup. During event processing, Nexel servlet first check whether this event is for the current JVM. Otherwise it will simply routes the ServletRequest to the corresponding JVM. This class provides methods for setting/getting JVM id and methods for routing requests to different JVMs.
Class com.nexaweb.server.ConnectionManager: This class maintains a Hashtable of ServletResponse instances. Each instances corresponds to a socket communication channel. Each instance can be uniquely located by the name of the thread that this instance belongs to. This class is necessary because the ServletResponse instance cannot be passed to event handlers and UI Objects directly. Event handlers and UI Objects should use this class to find the socket to send information to clients.
Class com.nexaweb.server.UIManager: This class is a replacement of javax.swing.UIManager. It loads a UI class corresponding to the client platform for painting.
Class com.nexaweb.server.PerformanceManager: This class provides methods for gauging the performance level of the current JVM. Methods in the class will be used for developing monitoring applications and making load balancing decisions. For example, if there performance level of the current JVM is below a certain threshold, the Nexel servlet may simply creates a new JVM for handling additional user applications.

Initial implementation of the above classes are shown in Appendix 1.

2.2.3.2.2 Nexel Java Foundation Classes

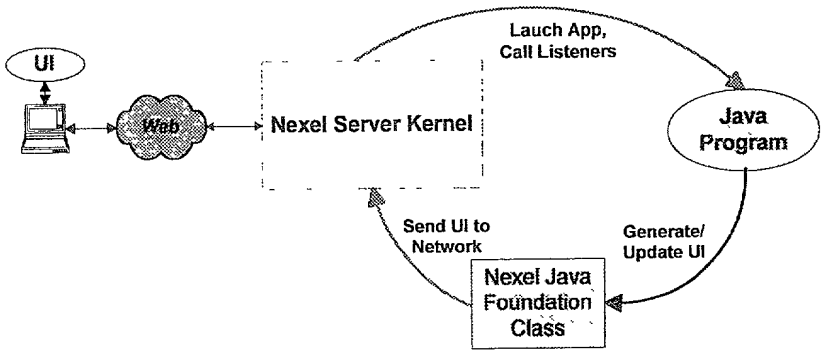


Figure 4: Nexel Java Foundation Class

Java Foundation Class, or JFC, is a loose collection of standard Java APIs for client-side graphics; graphical user interfaces (GUIs), and related programming tasks. They are the foundation classes in the sense that most client-side Java applications are built upon these APIs. More in detail, JFC is composed of four parts:

1. AWT. The Abstract Window Toolkit initially introduced. AWT is the foundation of Swing.

2. Swing. Swing is an advanced GUI toolkit written in pure Java.

3. Java 2D: this is a collection of classes offer two-dimensional graphics.

4. Printing. Both AWT and Swing offer support for printing text and graphics.

5. Data transfer, this includes cut &paste and drag drop.

Nexel is an enterprise application presentation platform that deliver application user interface to any device over any network connection, without modifying existing Java applications. Java applications typically build their UI on top of Java Foundation Class. Nexel modifies the behavior of JFC by replacing part of the implementation with its own implementation. This modified JFC is called "Nexel Java Foundation Class". Instead of painting UI on local computer screen, Nexel Java Foundation Class directly sends UI commands and directives to the client device over a network connection. However, Nexel Foundation Class keeps exact the same interface as Java Foundation Class. Namely, for any method in any class, they have the same signature. The only difference is the implementation of these methods. As a result, existing Java applications built using JFC can run as it is without modification. Developers can develop applications to run on top of Nexel without the need of learning anything new.

In the initial implementation, Nexel aims to support Swing and/or Java 2D based Java applications, but not AWT-based GUI applications. Nexel also plans to support printing and data transfer.

2.2.3.2.3 Technical Approach to implement NJFC

JFC is implemented by following a Model-View-Controller architecture. Each JFC component has a Model, a view and a controller. For example, for the JButton component, its model is AbstractButtonModel, its controller is JButton and its view is JbuttonUI (depends on the look & Feel). Sometimes the controller is combined with the model into one class.

To successfully replace JFC, for each component, we need to re-implement:

1. The component itself (the controller). For example, JButton needs to be re-implemented to achieve three things: a) all event listeners will be added to the Application instance instead of storing within the JButton class itself. Nexel Client Kernel will only send message to Nexel Server if and only if there is an event listener registered for a particular kind of event. So when an event listener is added, a directive indicating that Nexel Server is interested in such event should be sent to the client. 2). Actions for updating/painting the JButton should be sent to UIManager and further sent to the client device; c). When this JButton is created, assign it a unique component ID and store it in the Application Instance.

2. It's UI Class (Viewer). For each client platform, a view class needs to be implemented. This view class will be loaded if the client platform matches. This dynamic loading is achieved by UIManager. UIManager needs to check the client platform and loads the view class for that client platform. For example, for JButton, we need to implement WinJButtonUI, MacJButtonUI,XwinJButtonUI, and WinCEJButtonUI.

   The functionalities of this UI class are: a). to generate and send UI description of this component to Nexel Client Kernel via the ServletResponse object. The UI description is platform-dependent. It is a protocol that needs to be agreed between the Client Kernel and the UI class on the server side. b). to update UI on the client machine according server-side instructions. The update is accomplished by generating and sending a message describing what and how to update the component UI.

For painting, whether the target output is the client machine display device or a remote printer needs to be taken into design consideration.

For more information, see Design Specification "Nexel Client Kernel" .

3. The existing JFC Model classes are used in Nexel Java Foundation Class without modification.

A list of Swing components that we need to replace is shown in Figure 5.

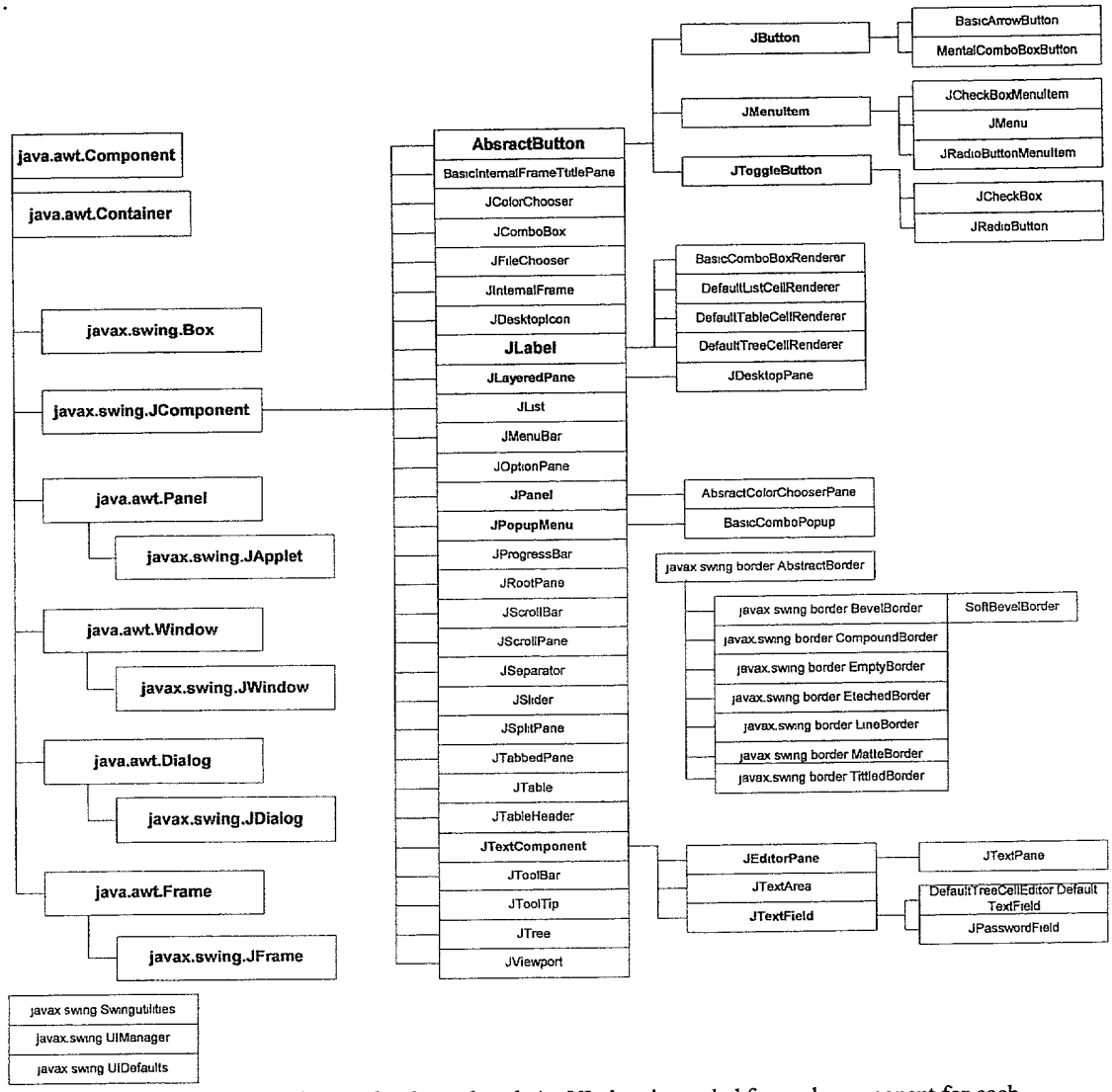


Figure 5: Swing Components that need to be replaced. An UI class is needed for each component for each client platform.

New Components	
Component	**Description**
JRadioList	List Box with Radio buttons for each item
JCheckList	List Box with Check buttons for each item

JDate	Date editing Control
JCalendar	Calendar Control
JStatusBar	Status Bar Control
JValidation	Runs validation on control values on the client side
	New Dialogs
JPrintDialog	Printer selection dialog

2.2.3.2.4 Nexel Layout Manager

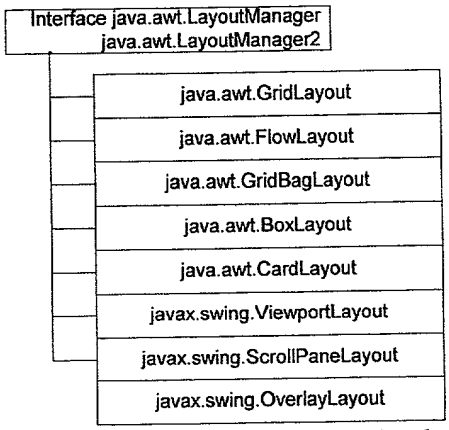


Figure 6: JFC/AWT Layout Managers that needs to be replaced

Figure 6 shows a list of layout managers that need to be replaced. Nexel Layout Managers should be adaptive to client display characteristics.

A deployment tool/design tool for laying out GUI components for different client devices will be provided. See "6 Nexel Client Display Engine" for more information.

2.2.3.2.5 Nexel Network Engine

Nexel Network Engine is packaged into one package com.nexaweb.server.net. This package is responsible for direct synchronous communcation with clients and other Java Virtual Machines that Nexel is running.
Class com.nexaweb.server.net.NetEngine:
This is one of the two entrances of the Nexel Server (The other one is via a Java Servlet Engine). This class creates instances of NexelServerSocket, each of which listens to a pre-determined port. It instantiates com.nexaweb.server.Nexel if it has not been instantiated;
Within a single JVM, only one instance of Main is allowed. If this JVM is started by another Nexel from a different JVM, this class's main method will be called with arguments that gives the IP address and port number corresponding to the calling JVM. Upon this JVM is started, a message should be sent to the calling JVM notifying "ready" status.
Class com.nexaweb.server.net.NexelServerSocket:
This class extends java.net.ServerSocket. The instance of this class listens to a specific port. Whenever it accepts a connection, it creates a new thread and lets the new thread (an instance of SocketHanlder) handles that connection. Then it returns immediately to listening to that port.
Class com.nexaweb.server.net.SocketHandler:
This class extends java.lang.Thread. The instance of this class reads/writes to a socket connection. When reading, it formats the information into an instance of NexelServletRequest and wraps this socket connection into an instance of NexelServletResponse. Passes both objects to the "service()" method in com.nexaweb.server.Nexel Servlet.

Class com.nexaweb.server.net.NexelServletRequest:

This class extends javax.servlet.ServletRequest. It basically wraps information coming from a socket connection into a ServletRequest Object so that they can be accessed using Java Servlet API.

Class com.nexaweb.server.net.NexelServletResponse:

This class extends javax.servlet.ServletResponse. It basically wraps a socket connection into a ServletResponse Object so that they can be accessed using Java Servlet API.

2.2.3.2.6 Additional Classes

The following classes are planned to be replaced:

1. Graphics 2D:

AWT provides basic graphics functionalities through instances of java.awt.Graphics. JFC further extends this by extending java.awt.Graphics into java.awt.Graphics2D. These two interfaces are responsible for all the drawing actions.

We need to provide Nexel implementation of java.awt.Graphics and java.awt.Graphics2D interfaces. Whenever method getGraphics() is called(This method is from java.awt.Component, java.awt.Image, java.awt.PrintJob and javax.swing.Jcomponent), we should return an instance of Nexel implementation of java.awt.Graphics2D. This instance should route all the drawing activities to Nexel Client Kernel.

Both AWT and Graphics2D provide additional classes such as java.awt.Rectangle and the entire java.awt.geom package for manipulating 2D graphics objects (such as affine transformation, setting/getting attributes and so on). These classes do not need to be modified.

2. Printing

AWT providing printing capability by java.awt.PrinterJob. JFC provides printing capability by offering a new package "java.awt.print". We plan to enable JFC-based printing.

As a result, java.awt.print.PrinterJob needs to be re-implemented: a). Its static method "getPrinterJob()" needs to be re-written so that a PrinterJob that rerepsents the client machine printer is returned; b). Methods pageDialog() and printDialog() should open dialogs on client machine; c). Method print() should create an instance of java.awt.Graphics2D according to the characteristics of the client machine printer, and pass this instance as an argument for calling "print()" method in the pageable or printable object. Every drawing action happening to the Graphics object should be routed to the client machine for printing.

3. Data Transfer

Data transfer refers to the ability of an application to transfer selected data I a variety of ways such as Cut&Paste and Drag&Drop. Nexel Platform plans to support both.

We plan to support data transfer among different applications on the same client machine. These applications include Nexel-based applications and other native applications. This feature will be enabled by default and its implementation is dependent on the native system.

Java provides two package called "java.awt.datatransfer" and "java.awt.dnd" . We need to selectively implement classes in these packages to enable data transfer on remote client machine. Details need to be further studies.

4. Accessiblity

This needs to be furher studied. In the short term, we delegate accessibility to the native operating system on the client machine.

5. java.awt.Toolkit.

This class needs to be re-implemented based on Nexel's distributed presentation model.

The following classes are planned to be added in addition to those offered by Java API:

1. Validation classes.
2. Other support classes.

2.2.3.2.7 Monitoring and Administration UI

The Monitoring and Administration UI is responsible for displaying the information tracked by the Monitoring and Administration Service. This UI will be developed using Java and JFC. Nexel server will be used to web enable this application. The application will contain main screen, which will have a table

view displaying all the information. There will be a filter window to filter the entries in the table. The filter should be available for all the information types. This application should also be able to display the active screen in the application.

2.2.3.2.8 Thread-based Computing

Nexel is based on a thread-based computing model that every request for the client machine is handled in its own thread. Once the processing is finished, that thread will simply be destroyed or recycled for other usage. Nexel does not maintain a constant process or even a constant thread for an application during its entire session. This is the computing model used in Java Servlet and has proven to be very scaleable. However, for efficiency reasons, Swing components are not designed to be thread safe. This means that Swing components should be manipulated by a single thread at a time. Nexel Platform needs to pay special attention to this issue during design time:

1. Client-side initiated threads. When a client request is received, either the Java Servlet Engine or Nexel Network Engine will allocate a thread for handling this request. Once the request is processed, this thread will be freed. However, for different client requests, if they belong to the same application, they need to be processed sequentially though they are in different threads. An event que for each application instance needs to be maintained.
2. Server-side initiated thread. There are times that developers need to update UI in response to some kind of eternal event,such as a response from a server that arrives in a separate thread. To accommodate these situations, Swing provides two utility methods that allow developers to ask the event queue to run arbitrary code. These methods are SwingUtilites.invokeLater() and SwingUtilities.invodeAndWait(). Nexel platform needs to re-implement these methods since Nexel does not use the standard JVM event queue thread.

2.2.3.9 Performance.

Nexel has to provide application delivery at acceptable performance. The time is spent in following actions while running an application under Nexel environment.

a. Sending instructions from client to server to launch an application. This time should be same as other web application
b. Launch Application Instance. Attempt should be made to optimize this time, as it may be significant. One way to reduce this is to launch each application as a different thread rather than a new process with new JVM. This poses certain restriction on having static variables.
c. Extraction of UI. This time significantly depends on how the Component Framework is implemented. It should be very optimized to reduce the time it needs to record its UI.
d. Transferring UI to client. This time should be same as other web applications.

Even though it is very difficult to define the parameters at this time, the performance and scalability should be very close to HTML applications.

2.2.3.10 Scalability.

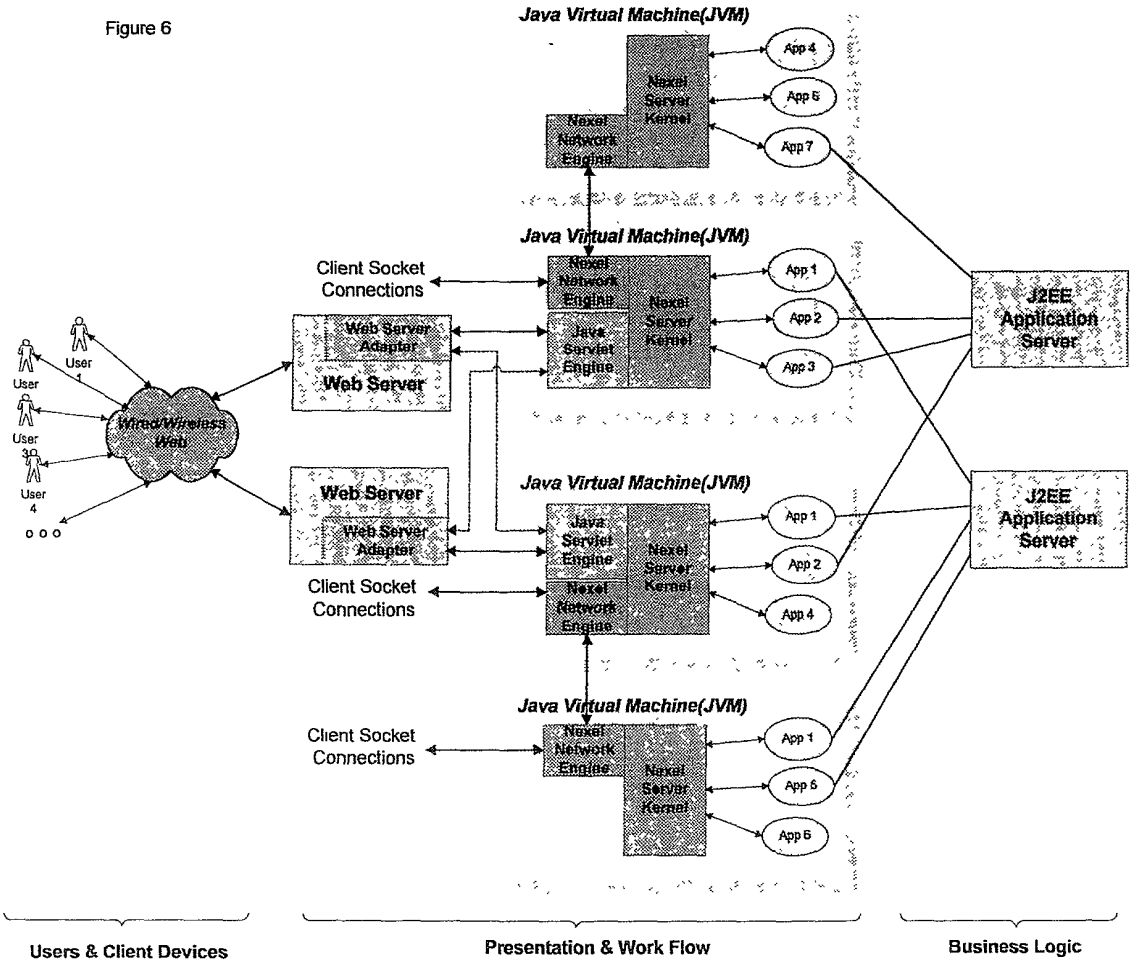
Nexel has to provide highly scalable applications. The server should be capable of running many application instances on server. Every application that is launched consumes certain amount of memory and the design should provide for ways to reduce this consumption. The ways to reduce this consumption are

a. Serializing the application components to the disk. This frees up memory for new instances to be launched. This at the same time will effect in slower performance.
b. Allowing application to discard lot of component model memory, which is not needed. For example in a tree control, if the application only concern about the selected item, which could be received from the client, then it can discard the memory for other tree items.
c. Pooling many JVM or computers into a Server Farm. Nexel Server could run on many machines and it could use different machine to launch many instances.

d. Adding more web server machine to Web server installation. This feature is found in many Web Servers (e.g. IIS) and could be utilized.

Figure 6 shows an enterprise-scale deployment of Nexel platform and its role in enterprise computing. Nexel Network Engine links different Nexel instances together and forms a high performance-computing environment with unlimited scalability. The diagram has two components

Figure 6



2.2.3.11 Restartability.

The applications run on the server machine should be accessible as long as they are running. If for any reason client breaks connection with the server, the same user which started the application should be able to recreate its UI. Server side application has to maintain information to be able to recreate UI. Some of the

changes that user did on client side may not be recreatable. The client kernel also reuse some of its cahced information inorder to recreate the screen.

2.2.4 Nexel Client Kernel.

Nexel Client Kernel provides functionality to play the application UI on the client machine. It receives the instructions from the NexaWeb UI server for creating and updating the applications UI. It captures all the events and provides default response to events that are not handled by application. Applications could decide to handle events on client side or server side depending on which Kernel will either execute specified code on client side of the application or notify the event on server side of the application. It will also wait for response from server and update the UI when asked. It also provides caching of an application UI on the client side

2.2.4.1 Nexel Client Kernel Platforms.

Client kernel needs to be implemented on many platforms that play an application UI. For Release 1.0 only three platforms are targeted.

Platform	Technology	Language for development
Internet Explorer	Active –X component	C++
	DHTML/JS code. Lot of this code exists today. Should be finished and tried out	DHTML/JS
Netscape Browser	Plugin	C++
Windows CE	Proprietary player needs to develop.	C++

The same code base is expected for Active-X component, Netscape Plugin and Windows CE player. These players will be developed using C++ language and so the code base should be same. On windows platform ATL Windows classes should be used to implement

2.2.4.2 Nexel Client Kernel Architecture.

The core functionality of the Client Kernel is to provide a rich Component Framework to create rich user interface. It creates instances of these components on the user's display and monitor events. The architecture for Client Kernel is shown below.

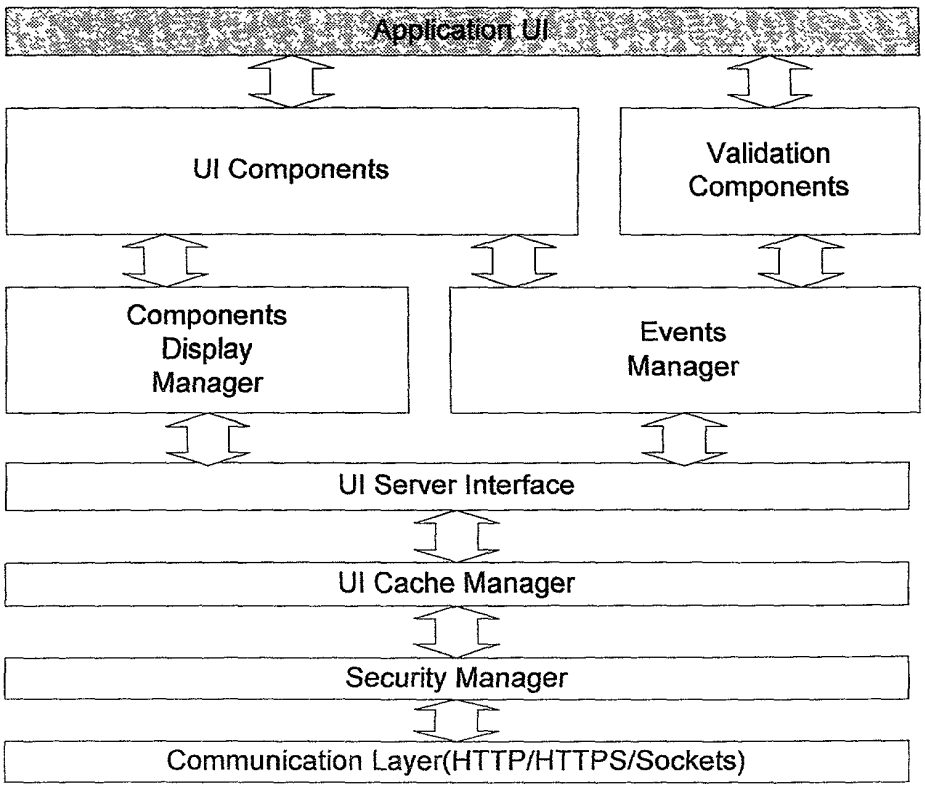


Figure 1 : Nexaweb Client Kernel Architecture

UI Components	This is a widget set that is implemented in the Kernel. It includes various widgets like button, window, tree etc.
Validation Components	These are components provides functionality for simple validation of data. These components are invoked on specified events and they execute specified validation rules
Components Display/Update Manager	This takes care of creating UI Components and updating them whenever necessary
Cache Manager	This manages Client Kernel Caching as well as application UI caching on the client side. On Windows CE Platforms it should bring up an application even though it is not connected to the server.
Event Manager	Event manager handles all the events
UI Server Interface	This module convert the packets received from the server into various UI and event monitoring commands.
Security Manager	Handles security for communicating with the server. This determines the protocol and encryption that will be used.
Communication Layer	This layer provides services for communicating with the server.

2.2.4.3 Security Manager.

The Security Manager handles security for the communication records going to and from server to client. Standard security is used when a Web Server is involved. The HTTPS protocol is used to handle security. For socket-based communication, our own security mechanism will be implemented. Evaluate third party products to do this.

2.2.4.3 UI Cache Manager.

The Cache Manager has two purposes:

- To cache the Client Kernel itself. The Kernel detects that there is a newer version of the Kernel available on the server and it downloads and runs that Client Kernel. For browser-based implementations, its own caching mechanism can be used. For non-browser based implementations such as Windows CE, it will have to be implemented.
- To cache the application UI. An application name and its version will be used for caching. The code detects application versions on the server and client and if they match, it will not download its UI from the server. The server can disable the caching. The different windows that are opened during different events will also be cached. This feature can be used to play the UI on a client even if there is no connection to the server.

The cache manager uses a cache directory to save files. This directory will be identified by the Registry key on the windows platform:

HKEY_CURRENT_USER\Software\NexaWeb\Nexel\Cache Directory

Environment variable on other platforms:

NEXEL_CACHEDIR

The directory structure used for caching should look like this:

+ HOME
  + Application Name
    + Version
      - File1
      - File 2
      - EventWindow.map

The directory contains a file EventWindow.map that keeps track of an event and the window that was opened due to the event.

2.2.4.3 UI Record Format.

The server sends the UI information to the client using this record format. Records are written in XML and packaged in text and transmitted over http protocol. This format may need tuning and can be changed to a compact binary format. The XML format is implemented, at least for testing purposes. The following principles are implemented:

- Each application instance has an assigned identification number.
- Each window is identified with an identifier from the server. This identifier is used in later communications for event handling and property changes.
- Every control implements properties, methods and events of its base class. Window class is the base class for all the controls.
- Every control is identified by a class name. This name is identified by the Widget name in UI Components table.
- The control's properties are described after identifying a window.
- The events that are to be handled are described later on.

Applications properties are listed in following table.

Application	Properties

Name	Description	Structure/Possible Values	XML Examples
Id	Identifier for App instance		id=123
Name	Application Name		name="MyApp"
Version	Application Version		version="1.0"
Cache	Use Caching	true,false	cache=true

A typical record may look like this:

```
<nexawebapp id=123 name="myapp" version =1.0 cache=true>
        <window id=1>
                <bkcolor>yellow</bkcolor>
                <text>My First Application</text>
                <toolbar id=2>
                        <button id=3 command=100>
                                <text>New</text>
                                <image>images/new.gif</image>
                        </button>
                        <button id=4 command=101>
                                <text>Exit</text>
                                <image>images/new.gif</image>
                        </button>
                </toolbar>
                <oncommand id=100></oncommand>
                <oncommand id=101></oncommand>
        <window>
</nexawebapp>
```

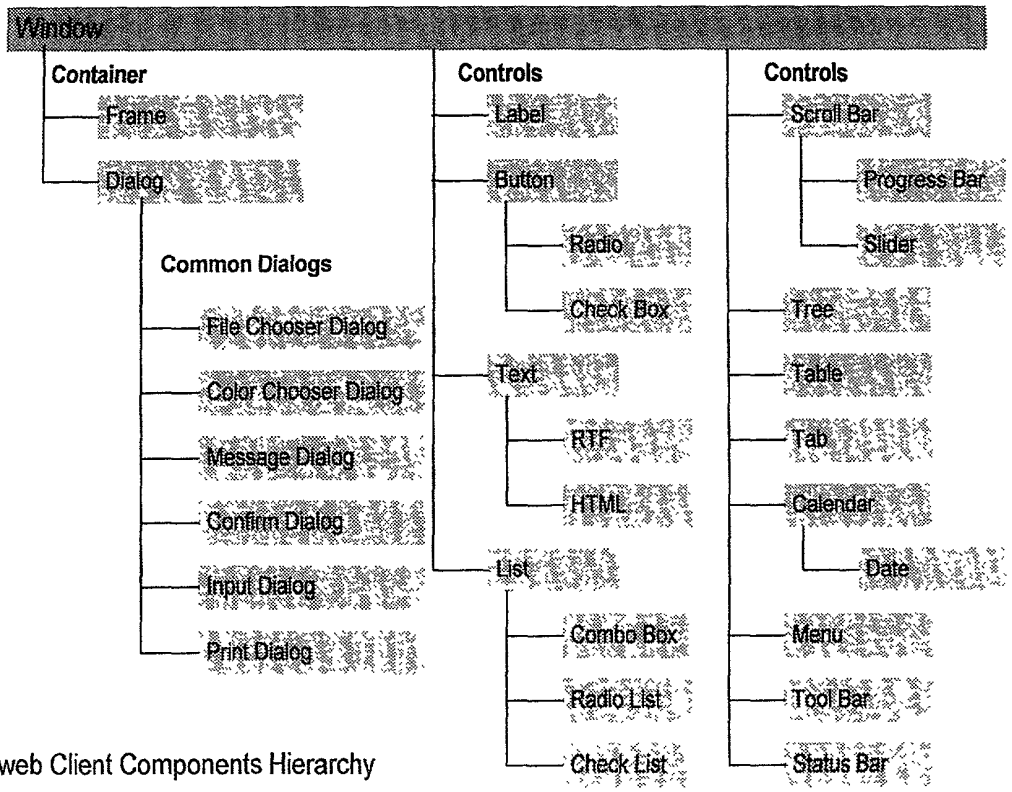
2.2.4.3 Event Record Format.

The Event Record Format specifies how the event records are passed back to the server. Events are passed back with application ID and a Window ID. It also collects the properties that are specified to be collected.

A typical event record may look like this:

```
<nexawebapp id=123>
        <window id=1>
                <oncommand id=100></oncommand>
        <window>
</nexawebapp>
```

2.2.4.3 Client Component Hierarchy.



Nexaweb Client Components Hierarchy

The following table displays the UI Components that are supported. Each Control supports its properties, methods and events. The following section describes each control that is implemented by the Client Kernel.

Components		
Widget Name	**Window Equivalent**	**Description**
Component	Window	Basic Window container
Dialog	Dialog	Dialog container
Frame	FrameWnd	MDI Frame container
Label	Static	Caption or label control
Push Button	Button	Push Button Control
Radio Button	Button	Radio Button Control
Check Button	Button	Check Button Control
Text	Edit	Edit/Text Control
RTF	RichEdit	Rich Text Edit
HTML	DHTMLEdit	HTML control
List	ListBox	List Control
Combo Box	ComboBox	Combo Box control
Radion List		List Box with Radio buttons for each item
Check List		List Box with Check buttons for each item
Progress Bar	ProgressCtrl	Progress Bar Control
Slider	SliderCtrl	Slider Control
Tree	TreeCtrl	Tree Control

Table	ListCtrl	Table Control
Tabbed Dialog	Tab	Tabbed Dialog
Date	DateTimePicker	Date editing Control
Scroll Bar	ScrollBar	Scroll Bar Control
Calendar	Calendar	Calendar Control
Menu	Menu	Menu
Tool Bar	ToolBar	Tool Bar control
Status Bar	StatusBar	Status Bar Control
Dialogs		
Message Dialog	MessageBox	Message dialog with OK and CANCEL button
Confirm Dialog	MessageBox	Confirm Dialog with YES,NO, CANCEL buttons
Input Dialog		Input Dialog with one edit control
File Dialog	FileDialog	File Selection Dialog
Color Dialog	ColorDialog	Color Chooser Dialog
Print Dialog	PrintDialog	Printer selection dialog

2.2.5 Nexel Communication Format.

Nexel provides many ways of communication between Client Kernel and Delivery Server. For each client type there could one implementation. In this release only one format is to be implemented. The communication happens in the UI record format and Event Record format. The following tables describe the properties, methods and event that each component could have.

Component			
Properties			
Name	Description	Structure/Possible Values	XML Examples
location	Window Location	Left, top, width, height Left or top = -1 locate anywhere Width or height = -1 freely size	location="10,10,-1-1"
bkcolor	Background color	Color Value	bkcolor=red
textcolor	Foreground color	Color Value	textcolor=green
state	Window state	minimized maximized normal	state=normal
minimizebox	Window has minimize box	True,false	minimizebox=true
maximizebox	Window hax maximize box	true,false	maximizebox=true
cancelbox	Window has cancel box	true, false	cancelbox=true
systemmenu	Window has system menu	true, false	systemmenu=true
visible	window is visible	true,false	visible=true
enabled	window is enabled	true,false	enabled=true
contexthelp	Window has help button	true,false	contexthelp=false

text	Window title/caption/text		text="My window"
border	Window border	none fixed dialog sizeable	border=none
focus	Window has focus	true/false	focus=true
font	Window font	see font description	
icon	Window Icon	value	icon="images/icon.gif"
cursor	Mouse pointer		cursor="images/hand.gif"
helpurl	Help URL		helpurl="help/mywindow.htm"
flash	Flash Window	true, false	flash=true
tooltiptext	Tool Tip Text		tooltiptext="Here is my tooltip"

Methods			
Name	Description	Parameters	XML Examples
print	Print the window		<print></print>
center	Center Window	window id	
redraw	Redraw Window		<redraw></redraw>
move	Move Window	left,top,width.height	<move>10,10,-1,-1</move>
show	Show Window		<show></show>
moveto	Move Drawing point		<moveto x=10 y=10> </moveto>
lineto	Draw Line		<pen>…</pen> <lineto x=10 y=10> </lineto>
circle	Draw Circle		<pen>…</pen> <brush>…</brush> <circle x=10 y=10 radius=20 fill=true> </circle>
rectangle	Draw Rectangle		<pen>…</pen> <brush>…</brush> <rectangle x=10 y=10 width=20 height=20 fill=true> </rectangle>
polygon	Draw polygon		<pen>…</pen> <brush>…</brush> <polygon points=10,10;20,20;30,30 fill=true> </rectangle>

Events			
Name	Description	Parameters	XML Examples
Onmousedown		button, x,y	<onmousedown button=left x=10 y=20></mousedown>
onmouseup			<onmouseup button=left x=10 y=20></mouseup>
Onmousemove			<onmousemove x=10 y=20></mousemove>
onclick			<onclick button=left x=10 y=20></onclick>
ondblclick			< ondblclick button=left x=10 y=20></ ondblclick >
onactivate	window is activated		<onactivate oldactiveid=122> </onactivate>
ondeactivate	window is deactivated		<ondeactivate newactiveid=133> </ondeactivate>
Oncontextmenu	context menu is requested		<oncontextmenu></oncontextmenu>
onfocus	window is getting		<focus oldfocusid=111> </focus>

	focus		
onkillfocus	window is loosing focus		<onkillfocus newfocusid=111> </onkillfocus>
onhelp	help is invoked		<onhelp></onhelp>
onkeydown	key is down		<onkeydown key="b" shift=true alt=false ctrl=true> </onkeydown>
onkeyup	key is up		<onkeyup key="b" shift=true alt=false ctrl=true> </onkeyup>
onkeypress	key is pressed		<onkeypress key="b" shift=true alt=false ctrl=true> </onkeypress>
onresize	window is resized		<onresize width=10 height=50> </resize>
onmove	window is moved		<onmove left=10 top=20> </onmove>
oncommand	Window recieves a command		<oncommand id=200> </oncommand>

Dialog			
Properties			
Name	Description	Structure/Possible Values	XML Examples
modal	Dialog is modal	true/false	modal=true
Methods			
Name	Description	Parameters	XML Examples
reset	reset the dialog		<reset></reset>
Events			
Name	Description	Parameters	XML Examples
submit	Submit dialog		<submit></submit>

Frame			
Properties			
Name	Description	Structure/Possible Values	XML Examples
splitdir	Split Direction	horz,vert	splitdir=horz
Methods			
Name	Description	Parameters	XML Examples
split	Split the frame	pane, no of panes	<split pane=0,1 panes=2></split>
attach	Attach Window	window	<attach windowid=100 > </attach>
Events			
Name	Description	Parameters	XML Examples

Label			
Properties			
Name	Description	Structure/Possible Values	XML Examples
Methods			
Name	Description	Parameters	XML Examples
Events			
Name	Description	Parameters	XML Examples

Push Button

Properties			
Name	Description	Structure/Possible Values	XML Examples
image	Image	image file name	image="images/yes.gif"
overimage	Image to be displayed when mouse is moved over		overimage="images/no.gif"
downimage	button image when pushed down		downimage="images/down.gif"
selectedimage	button image when selected		selectedimage="images/selected.gif"
imageloc	image location in reference to text	left,top,right,bottom	imageloc=top
imagealign	image alignment	left,right,center,top,bottom,vcenter	imagealign="top,center"

Methods			
Name	Description	Parameters	XML Examples

Events			
Name	Description	Parameters	XML Examples
oncommand	Button is pushed		<oncommand id=202> </oncommand>

Radio Button

Properties			
Name	Description	Structure/Possible Values	XML Examples
selected	radio button is selected	true,false	selected=true

Methods			
Name	Description	Parameters	XML Examples

Events			
Name	Description	Parameters	XML Examples

Check Box

Properties			
Name	Description	Structure/Possible Values	XML Examples
checked	button is cheked	true,false	checked=true

Methods			
Name	Description	Parameters	XML Examples

Events			
Name	Description	Parameters	XML Examples

Group Box

Properties			
Name	Description	Structure/Possible Values	XML Examples

Methods			
Name	Description	Parameters	XML Examples

Events			
Name	Description	Parameters	XML Examples

Text

Properties			
Name	Description	Structure/Possible Values	XML Examples
multiline	whether multiline text	true,false	multilne
textlimit	number of characters allowed	number	textlimit=10
editmask	mask to be used for editing	string	editmask="###.##.####"
datatype	type of data	string,number,date,currency,amount	datatype=date
outformat	output format	string	outformat="$%x"
align	Alignment	left,center,right	align=right

Methods			
Name	Description	Parameters	XML Examples
clear	Clear the text		<clear></clear>
cut	Cut the selected text to clipboard		<cut></cut>
paste	Paste from clipboard		<paste></paste>
copy	Copy the selected text to clipboard		<copy></copy>

Events			
Name	Description	Parameters	XML Examples
onchange	Text is changed		<onchange text="1234"></onchange>

RTF

Properties			
Name	Description	Structure/Possible Values	XML Examples

Methods			
Name	Description	Parameters	XML Examples

Events			
Name	Description	Parameters	XML Examples

HTML

Properties			
Name	Description	Structure/Possible Values	XML Examples
Methods			
Name	Description	Parameters	XML Examples
Events			
Name	Description	Parameters	XML Examples

List

Properties			
Name	Description	Structure/Possible Values	XML Examples
listitem	List item description. many records repeated	text,image,data	<listitem text="xyz" image="images/checked.gif" data="avalue" ></listitem>
selectedtext	Selected item text		selectedtext="xyz"
selecteddata	Selected item data		selecteddata="avalue"
itemcount	Number of items		itemcount=10
multiselect	Multiple selection Allowed	true,false	multiselect=true
sorted	List is sorted	true,false	sorted=false
Methods			
Name	Description	Parameters	XML Examples
additem	Adds a item	listitem	<additem text="xyz" data="abc"> </additem>
Events			
Name	Description	Parameters	XML Examples
selchange	Selection Changed		<selchange index=0 text="xyz" data="abc"> </selchange>

Radio List

Properties			
Name	Description	Structure/Possible Values	XML Examples
Methods			
Name	Description	Parameters	XML Examples
Events			
Name	Description	Parameters	XML Examples

Check List

Properties			
Name	Description	Structure/Possible Values	XML Examples
Methods			
Name	Description	Parameters	XML Examples
Events			
Name	Description	Parameters	XML Examples

Scroll Bar			
Properties			
Name	Description	Structure/Possible Values	XML Examples
direction	Progress direction	horz,vert	direction=vert
range	Range	low,high	range=100,200
pos	Position		pos=10
linestep	Step		linestep=10
pagestep	Page Step		pagestep=100
Methods			
Name	Description	Parameters	XML Examples
scroll	Scroll the bar by offset		scroll=-10
setscroll	Set the thumb		setscroll=100
Events			
Name	Description	Parameters	XML Examples
poschange	Position Changed		<poschange newpos=105> </poschange>

Progress Bar			
Properties			
Name	Description	Structure/Possible Values	XML Examples
smooth	Smooth Scrolling bar	true,false	smooth=true
step	Step		step=10
Methods			
Name	Description	Parameters	XML Examples
stepit	Advance		<stepit numofsteps=10> </stepit>
Events			
Name	Description	Parameters	XML Examples

Slider			
Properties			
Name	Description	Structure/Possible Values	XML Examples
Methods			
Name	Description	Parameters	XML Examples

Events			
Name	Description	Parameters	XML Examples

Calendar			
Properties			
Name	Description	Structure/Possible Values	XML Examples
daterange	Range in Calendar Control	from, to	<daterange from="1/1/99" to="2/2/00"></daterange>
date	Selected date		date="1/1/00"
Methods			
Name	Description	Parameters	XML Examples
Events			
Name	Description	Parameters	XML Examples
selchange	Date selection changed		<selchange newdate="1/1/00"> </selchange>

Date			
Properties			
Name	Description	Structure/Possible Values	XML Examples
inputmask	Input Mask		inputmask="mm/dd/yy"
outformat	Output Format		outformat="dd-mmm-yyyy"
Methods			
Name	Description	Parameters	XML Examples
Events			
Name	Description	Parameters	XML Examples

Tree			
Properties			
Name	Description	Structure/Possible Values	XML Examples
treeitem	Tree Item Structure		<treeitem parent=0 id=1 text="abcd" image="open.gif" openimage="close.gif" selectedimage="selected.gif" children=1 data="ext111" state=expanded> </treeitem>
selecteditem	Selected Tree Item		<selecteditem id ="1" text="abcd" data="ext111"> </selecteditem>
rightclickitem	Item where right clicked		<rightclickitem id ="1" text="abcd" data="ext111"> </rightclickitem>
Methods			
Name	Description	Parameters	XML Examples
insertitem	Insert Item		<insertitem> <treeitem>…</treeitem> </insertitem>
deleteitem	Delete Item		<deleteitem> <treeitem>…</treeitem>

			</deleteitem>
deleteall	Delete all items		<deleteall> </deleteall>
expanditem	Expand Item		<expanditem id=1> </expanditem>
selectitem	Select Item		<selectitem id=1> </selectitem>
sortchildren	Sort Children		<sortchildren id=1></sortchildren>
ensurevisible	Ensure Visible		<ensurevisible id=1> </ensurevisible>
editlable	Edit Label		<editlabel id=1> </editlabel>
setitem	Set Item		<setitem> <treeitem>…</treeitem> </setitem>
Events			
Name	Description	Parameters	XML Examples
begindrag	Begin Drag		<begindrag id=1> </begindrag>
enddrag	End Drag		<enddrag id=1 droptarget=5 > </enddrag>
itemexpanded	Item expanded		<itemexpanded id=1 text="xyz" data="as111"> </itemexpanded>
itemexpanding	Item expanding		<itemexpanding id=1 text="xyz" data="as111"> </itemexpanding>
selchanged	Selection Changed		<selechanged id=1 text="xyz" data="as111"> </selechanged>
selchanging	Selection Changing		<selechanging id=1 text="xyz" data="as111"> </selechanging>

Table

Properties			
Name	Description	Structure/Possible Values	XML Examples
tablecol	Table Header Column	see table column	<tablecol>…</tablecol>
tablerow	Table Row		<tablerow>…</tablerow>
tablecell	Table Cell		<tablecell>…<tablecell>
selectedrows	Selected Row		selectedrows=10,11,12,13
Methods			
Name	Description	Parameters	XML Examples
insertrow	Insert Row		<insertrow> <tablerow>…</tablerow> </insertirow>
deleterow	Delete Row		<deleterow row=1> </deleterow>
deleteall	Delete all Rows		<deleteall> </deleteall>
selectrow	Select Row		<selectrow row=1> </selectrow>
sort	Sort table		<sort column=1></sort>
ensurevisible	Ensure Visible		<ensurevisible row=1> </ensurevisible>
setrow	Set Row		<setrow row=1> <tablerow>…</tablerow> </setrow>
settablecell	Set Table Cell		< settablecell row=1 col=5> <tablecell>…</tablecell> </ settablecell >
insertcol	Insert Column		<insertcol after=1> <tablecol>…<tablecol> </insertcol>
deletecol	Delete Column		<deletecol col=1> </deletecol>
deleteallcol	Delete All Columns		<deleteallcol> </deleteallcol>

Events			
Name	Description	Parameters	XML Examples
begindrag	Begin Drag		<begindrag id=1> </begindrag>
enddrag	End Drag		<enddrag id=1 droptarget=5 > </enddrag>
selchanged	Selection Changed		<selechanged id=1 text="xyz" data="as111"> </selechanged>
selchanging	Selection Changing		<selechanging id=1 text="xyz" data="as111"> </selechanging>

Tabbed Dialog

Properties			
Name	Description	Structure/Possible Values	XML Examples
tab	Tab stucture	All button attributes and view attribute	<tab id=1 image="mytab.gif" text="abcd" view="abcd"> </tab>
selectedtab	Selected Tab		<selectedtab> <tab>…</tab> </selectedtab>

Methods			
Name	Description	Parameters	XML Examples
inserttab	Insert Tab		<inserttab> <tab>…</tab> </inserttab>
deletetab	Delete Tab		<deletetab> <tab>…</tab> </deletetab>
deleteall	Delete all tabs		<deleteall> </deleteall>
selecttab	Select Tab		<selecttab id=1> </selecttab>
settab	Set tab		<settab> <tab>…</tab> </settab>

Events			
Name	Description	Parameters	XML Examples
selchanged	Selection Changed		<selechanged id=1 text="xyz" data="as111"> </selechanged>
selchanging	Selection Changing		<selechanging id=1 text="xyz" data="as111"> </selechanging>

Status Bar

Properties			
Name	Description	Structure/Possible Values	XML Examples
pane	Pane	All button attributes	<pane id=0 text="For help press F1"> </pane>

Methods			
Name	Description	Parameters	XML Examples
addpane	Add Pane		<addpane> <pane>…</pane> </addpane>
setpane	Set Pane		<setpane><pane>…</pane> </setpane>

Events			
Name	Description	Parameters	XML Examples

Tool Bar

Properties			
Name	Description	Structure/Possible Values	XML Examples
tool	Tool coulbd be a button or other control	All control Attributes	<tool id=1> <button>…</button></tool>
docked	Which side docked	top,left,bottom,right,float	docked=float
moveable	Can be moved	no,canfloat,yes, candock	moveable=canfloat
Methods			
Name	Description	Parameters	XML Examples
addtool	Add Tool		<addtool after=1><tool>…</tool> </addtool>
settool	Set Tool		<settool><tool>…</tool></settool>
Events			
Name	Description	Parameters	XML Examples
oncommand	When tool is clicked		<oncommand id=100> </oncommand>

Menu

Properties			
Name	Description	Structure/Possible Values	XML Examples
menuitem	Menu Item	See menuitem description	<menuitem>…</menuitem>
layout	Layout	vert,horz	layout=vert
Methods			
Name	Description	Parameters	XML Examples
addmenu	Add Child Menu		<addchild after=1> <menuitem>…</menuitem> </addchild>
setmenu	Change Properties		<setmenu> <menuitem>…</menuitem> </setmenu>
Events			
Name	Description	Parameters	XML Examples
oncommand	When menu is clicked		<oncommand id=100> </oncommand>

Message Dialog

Properties			
Name	Description	Structure/Possible Values	XML Examples
message	Message		message="Test Message"
icon	Icon		icon="images/image.gif"
buttons	Buttons	ok,cancel	buttons=ok/cancel
return	Button Presses	ok,cancel	return=ok
Methods			
Name	Description	Parameters	XML Examples
Events			

Name	Description	Parameters	XML Examples
close	Dialog closed		<close return=ok></close>

Confirm Dialog

Properties

Name	Description	Structure/Possible Values	XML Examples
message	Message		message="Test Message"
icon	Icon		icon="images/image.gif"
buttons	Buttons	yes,no,cancel	buttons=ok/cancel
return	Button Pressed	yes,no,cancel	return=ok

Methods

Name	Description	Parameters	XML Examples

Events

Name	Description	Parameters	XML Examples
close	Dialog closed		<close return=ok></close>

Input Dialog

Properties

Name	Description	Structure/Possible Values	XML Examples
message	Message		message="Test Message"
icon	Icon		icon="images/image.gif"
buttons	Buttons	ok,cancel	buttons=ok/cancel
return	Button Presses	ok,cancel	return=ok
datatype	Data Type	string,date,number	datatype=date

Methods

Name	Description	Parameters	XML Examples

Events

Name	Description	Parameters	XML Examples
close	Dialog closed		<close return=ok text="acde"> </close>

Color Dialog

Properties

Name	Description	Structure/Possible Values	XML Examples
color	Color		color=red

Methods

Name	Description	Parameters	XML Examples

Events

Name	Description	Parameters	XML Examples
close	Dialog closed		<close return=ok color=black> </close>

File Dialog

Properties

Name	Description	Structure/Possible Values	XML Examples
startdir	Starting Directory		startdir="c:\"

filter	File Filter		filter="*.*"
file	Initial File		file="My File"
multifile	Allow Multiple file selection	true,false	multifile=false
existfile	Existing Files only	true,false	existfile=true
Methods			
Name	Description	Parameters	XML Examples
Events			
Name	Description	Parameters	XML Examples
close	Dialog closed		<close return=ok file="abc.gif;gdi.gif"> </close>

Printer Dialog			
Properties			
Name	Description	Structure/Possible Values	XML Examples
printer	Printer		printer="HP Laserjet III"
Methods			
Name	Description	Parameters	XML Examples
Events			
Name	Description	Parameters	XML Examples
close	Dialog closed		<close return=ok printer=" HP Laserjet III"> </close>

Validation			
Properties			
Name	Description	Structure/Possible Values	XML Examples
validateid	Control to validate		validateid=10
compareid	Control to Compare		compareid=15
valtype	Validation type	required, compare, range, expression	valtype=range
comparetype	Compare Type	equal, lessthan, greatherthan, notequal	comparetype="equal,lessthan"
message	Message when failed		message="Range validation Failed »
minvalue	Minimum Value		minvalue=10
maxvalue	Maximum Value		maxvalue=40
expression	Expression to check		expression="[0-9]{3}\s[0-9]{3}-[0-9]{4}"
Methods			
Name	Description	Parameters	XML Examples
Events			
Name	Description	Parameters	XML Examples
close	Dialog closed		<close return=ok printer=" HP Laserjet III"> </close>

Font			
Properties			
Name	Description	Structure/Possible Values	XML Examples
size	Size of Font		size=10
face	Font Face		face="Time Roman"
bold	Bold ?	true,false	bold=true
underline	Underline	true,false	underline=true
italic	Italic	true,false	italic=false
strikethrough	Strikethrough	true,false	strikethrough=false
textcolor	Text Color		textcolor=red
backcolor	Background Color		backcolor=yellow

Pen			
Properties			
Name	Description	Structure/Possible Values	XML Examples
size	Size of pen		size=10
shape	Shape of pen	square,round	shape=square
color	Color		color=red
style	Pen Style	solid,dash,dot, dashdot,dashdotdot	style=dash
insideframe	Drawing is inside bounding limits	true,false	insiderframe=true

Brush			
Properties			
Name	Description	Structure/Possible Values	XML Examples
style	Brush Style	solid,bidigonal, cross,diagcross, fdiagonal,horz, vert, image	style=dash
color	Color		color=red
image	Pattern Image		iamge="pattern.gif"

Structures Used in Components

Menu Item			
Properties			
Name	Description	Structure/Possible Values	XML Examples
text	Text		text="&File"
image	Image		image="images/file.gif"
imageloc	image location in reference to text	left,top,right,bottom	imageloc=top
imagealign	image alignment	left,right,center,top,bottom,vcenter	imagealign="top,center"
popup	Popup menu	true,false	popup=true

visible	Visible	true,false	visible=true
enabled	Enabled	true,false	enabled=false
checked	Checked	true,false	checked=false
hint	Menu hint		hint="Open a File"
data	Menu data		data="abcd"
id	Command id		id=100
hotkey	Hot Key		hotkey="ctrl+o"
align	Alignment	left,top,right,bottom	align=left

TableCell			
Properties			
Name	Description	Structure/Possible Values	XML Examples
row	Row		row=10
col	Column		col=10
text	Column text		text="abcd"
image	Column Image		image="images/attach.gif"
bigimage	Big Image		bigimage="images/bigattach.gif"
data	Cell Data		data="celldatar"

Table Header			
Properties			
Name	Description	Structure/Possible Values	XML Examples
col	Column		col=10
text	Column text		text="abcd"
image	Column Image		image="images/attach.gif"
data	Cell Data		data="coldatar"
width	Cell Width		width=100
sortedon	Sorted on this column	true,false	sortedon=true

Table Row			
Properties			
Name	Description	Structure/Possible Values	XML Examples
tablerow	Row		<tablerow row=10 data="abcd" > <tablecell>...</tablecell> <tablecell>...</tablecell> </tablerow>

2.2.6 Layout Manager

The Layout Manager allows adjusting the screens produced by a Java Application. This adjustments may be needed depending on client type. This application has to be developed using JFC and Java. This tool will load screens from application and will allow changing some of their properties. Once the screen has been

adjusted it will save a template into a file. The developer could decide the file format. A Java class for loading the file needs to be provided so that applications can use it.

2.2.7 Code Analyzer

The Code Analyzer will analyze existing Java applications and pin point the problems running that application under Nexel environment. The programmer can make those changes and recompile the application and deploy it. At this point in time, limitations are not known in great detail. So a rule based engine needs to be developed. This could be a command line tool, which analyzes application source code and finds problems and suggest alternatives, much like a compiler.

Appendix 1. Nexel Core Classes Initial Implementation

Class com.nexaweb.server.ConnectionManager

```
package com.nexaweb.server;

import java.lang.*;
import java.lang.reflect.*;
import java.util.*;
import java.io.*;
import java.text.*;
import java.awt.event.*;
import java.awt.*;

import javax.servlet.*;
import javax.servlet.http.*;

public class HttpManager {

protected static Hashtable threadList=new Hashtable();

 public  HttpManager() {return;}


public synchronized static void put(String tname, ServletResponse httpResponse)
{
  threadList.put(tname,httpResponse);
}

 public synchronized static void remove(String tname)
{
   System.out.println("Removing entry for "+tname);
   threadList.remove(tname);
}

 public synchronized static void remove()
 {Thread th=Thread.currentThread();
  String name=th.getName();
  threadList.remove(name);
}

 private static Object getConnection(String tname) {
  System.out.println("Get connection:threadList="+threadList+",name="+tname);

  Object o=threadList.get(tname);
  System.out.println("Get connection:threadList="+threadList+",connection="+o);
  return o;
}

public static Object getConnection() {
 Thread th=Thread.currentThread();
 String name=null;
 if(th instanceof AppServiceThread) {
   /**
   *in this case, the connection is stored into HttpManager in a parent thread
   *, and the retrieving happens in a child thread
   */
     AppServiceThread ath=(AppServiceThread)th;
     name=ath.getParentThreadName();
    System.out.println("this is an AppServiceThread: parentName="+name);
}
 else name=th.getName();
 System.out.println("Tring to get connection by thread name="+name);
 Object oo=getConnection(name);
 /*Object po=oo;
   for(int i=0;true;) {break;
         if(po==null) break;
         System.out.println("Class is. "+po.getClass().getName()+"\n");
          po=po.getClass().getSuperclass();
     }
  */

 try {
 System.out.println("HttpManager:get connection
="+oo+",oo,class="+oo.getClass().getName());
 }catch(Exception ee) {System.out.println("Exception in HTTPManager:"+ee);}
 return oo;
}

 }
```

Class com.nexaweb.server.Nexel

```
import java.io.*;
import java.text.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import com.nexaweb.server.*;


/**
 *Nexel Application Presentation Server
 *via Java Servlet interface
 */
public class Nexel extends HttpServlet {

   public void doGet(HttpServletRequest request,
               HttpServletResponse response)
     throws IOException, ServletException
   {

      PrintWriter out = response.getWriter();

     //App Launching Format: http://hostname:port/
servletname?appName=app1&user=user1;

     String appName=request.getParameter("appName");
     String user=request.getParameter("user");

     //App Messaging Format: http://hostname:port/
servletname?appid=appid&ctrlid=cid&key=key&eventid=eid&evparam=param
     String eid=request.getParameter("eventid");
     String appid=request.getParameter("appid");
     String cid=request.getParameter("ctrlid");

     System.out.println("Servlet
Path="+request.getServletPath()+",servername="+request.getRemoteAddr()+",
port="+request.getServerPort()+",pathinfo="+request.getPathInfo()+",URI="+re
quest.getRequestURI()+",path translated="+request.getPathTranslated());

     System.out.println("Request="+request.toString());
     System.out.println("Do post/Get.
eventid="+eid+",appid="+appid+",ctrlid="+cid);
     response.setContentType("text/html");

     out.println("<html>");
     out.println("<body bgcolor=\"lightblue\">");
     out.println("<head>");

     out.println("<title> Nexel Application Deliver Platform </title>");
     out.println("</head>");
     out.println("<body>");
     //out.println(rb.getString("requestparams.params-in-req") + "<br>");

    if(eid!=null&&eid.length()>1&&appid!=null&&appid.length()>0) {
      dispatchEvent(request,response,appid,cid,eid);
      return;
    }

     if (appName != null && user != null) {

      out.println("Application="+appName);
      out.println("user="+user);
      launchApp(request,response,appName,user);

    } else {
      //out.println(rb.getString("requestparams.no-params"));
    }
```

Class com.nexaweb.server.Nexel

```
     out.println("<P><h1>Nexel Application Delivery Platform Demo</h1>");
     out.print("<form action=\"");
     String action="http://
"+request.getServerName()+":"+request.getServerPort()+request.getRequestU
RI();

     out.print(action+"\" ");
     out.println("method=POST>");
     out.println("AppName");
     out.println("<input type=text size=20 name=appName>");
     out.println("<br>");
     out.println("User");
     out.println("<input type=text size=20 name=user>");
     out.println("<br>");
     out.println("eventid");
     out.println("<input type=text size=20 name=eventid>");
     out.println("<br>");
     out.println("appid");
     out.println("<input type=text size=20 name=appid>");
     out.println("<br>");
     out.println("Control");
     out.println("<input type=text size=20 name=ctrlid>");
     out.println("<br>");
     out.println("<input type=submit>");
     out.println("</form>");

     out.println("</body>");
     out.println("</html>");
  }

public void doPost(HttpServletRequest request,
            HttpServletResponse response)
   throws IOException, ServletException
  {
    doGet(request, response);
  }

 protected void dispatchEvent(HttpServletRequest request,
            HttpServletResponse response,String appid,String cid,String eid)
    throws IOException, ServletException
  {
    System.out.println("Dispatching event: appid="+appid+",eventid="+eid);

com.nexaweb.server.EventManager.dispatchEvent(request,response,appid,cid,
eid);
     System.out.println("Finished Dispatching event:
appid="+appid+",eventid="+eid);
  }

 protected void launchApp(HttpServletRequest request,
              HttpServletResponse response, String appName,
              String userName)
  {
   System.out.println("Launching application : "+appName);

   Thread thread=java.lang.Thread.currentThread();
   String tname=thread.getName();
   System.out.println("Working... Current thead name ="+tname);

   Vector argsV=new Vector();
   for(int i=0;i<100;i++) //maximum arguments is 100
   {String arg=request.getParameter("apparg"+i);
   System.out.println("arguments="+arg);
    if(arg!=null) argsV.addElement(arg);
    else break;
   }

   String[] args=new String[argsV.size()];
   argsV.copyInto((Object[])args);
   Application app=new com.nexaweb.server.Application(appName,args);


   app.setBaseURL("http://
"+request.getServerName()+":"+request.getServerPort()+request.getRequestU
RI());
   System.out.println("Application Base URL="+app.getBaseURL());

   HttpManager.put(tname,response);
   System.out.println("HTTP
response="+response+",class="+response.getClass().getName());
   AppManager.addAppThread(tname,app.getAppId());


System.out.println("****************class="+response.getClass().getName()+",
cf reponse?"+(response instanceof ServletResponse));
   try {
     app.start();
     System.out.println("Started Application ...");
   }catch(Exception ee) {
     System.out.println("Nexaweb Application start exception: "+ee);
   }
   HttpManager.remove(tname); //remove it after done
   AppManager.removeAppThread(tname);

   //SimpleTest.main();
  }
}
```

Class com.nexaweb.server.AppManager

```
package com.nexaweb.server;

import java.lang.*;
import java.lang.reflect.*;
import java.util.*;
import java.io.*;
import java.net.*;
import java.rmi.server.*;
import java.awt.*;

import javax.servlet.*;
import javax.servlet.http.*;

/**
 * class to hold all application instances
 */
public class AppManager extends java.lang.Object {

protected static int appCount=0;
protected static Hashtable appTable=new Hashtable();
protected static Hashtable appThreads=new Hashtable();

public AppManager() {return;}

public static String createNewAppId() {
    appCount++;
    return("ac"+appCount+System.currentTimeMillis());
    }

public synchronized static void addApp(Application app)
{ String key=app.getAppId();
  appTable.put(key, app);
}

public synchronized static void addAppThread(String tname,String appId) {
 appThreads.put(tname,appId);
}

public synchronized static void removeAppThread( String tname) {
 appThreads.remove(tname);
}

public synchronized static void removeApp(String appId)
{  Object app=appTable.get(appId); app=null;
   appTable.remove(appId);
   appThreads.remove(appId);
}

public static Application getApplication(String appId)
{ if(appId==null) return null;
  Object app=appTable.get(appId);
  return (Application)app;
}

public static Application getApplication()
{
  //System.out.println("Get application");
  Thread th=Thread.currentThread();
  String tname=th.getName();
  //System.out.println("Thread name="+tname);
  String appId=(String)appThreads.get(tname);
  //System.out.println("Application ID="+appId);
  return getApplication(appId);
}

/**
 *a helper method to create a unique component ID for each compoent
 *(The uniqueness is only within the scope of the application)
 */
public static String getUniqueComponentID() {
  Application app=getApplication();
  if(app==null) return "Can not find application";
  return app.getUniqueComponentID();
}
}
```

Class com.nexaweb.server.Application

```
package com.nexaweb.server;

import java.lang.*;
import java.lang.reflect.*;
import java.util.*;
import java.io.*;
import java.awt.*;

/**
 * Class to hold application information
 * This is necessary since we change the threading model of java programs.
We don't maintain
 * a main thread for each application any more. Our model is a service-based
model, each service
 * is served in its own thread. Once the service finished, the thread will die.
 * In order to keep different piece of an application together, we create an
Application class
 * to achieve that, since different piece of the application will be handled by
different threads.
 *
 */
public class Application extends java.lang.Object {
 protected String appName;
 protected String[] arguments;
 protected String appId;
 protected int componentCount=0;
 protected Hashtable listenerTable=new Hashtable();
 protected ThreadGroup group;
 protected String baseURL="";

 /**
  *A table to hold all the GUI components that belong to this application
  */
 protected Hashtable ctrlTable=new Hashtable();

 /**
  *A table to hold all other non-GUI components. This is needed when some
information
  * needs to be maintained during the entire application process, though the
thread
  * that created such information may have died.
  *
  *Each application instance is associated with one thread group. All threads
that belong
  * to this application belong to this thread group. This thread group has the
same name as
  * appId.
  */
 protected Hashtable dataTable=new Hashtable();

 public Application(String name, String[] args)
   {String tname=Thread.currentThread().getName();
    appId=AppManager.createNewAppId();//name+System.currentTimeMillis();
    //group=new java.lang.ThreadGroup(appId);
    group=Thread.currentThread().getThreadGroup();
    System.out.println("The thread group name for application
"+name+"="+group.getName());
    appName=name;
    arguments=args;

    AppManager.addApp(this);
    }

 public String getAppId() {
      return appId;
 }

 public  String getUniqueComponentID() {
    componentCount++;
    return "ctrl"+componentCount;
 }

 public void setAppId(String id) {appId=id;}

 public int getNumberOfComponents() {return componentCount;}

 public ThreadGroup getThreadGroup() {
    return group;
    }

 public String getThreadGroupName() {
    return group.getName();
 }

 public void setBaseURL(String s) {baseURL=s;}
 public String getBaseURL() {return baseURL;}
```

Class com.nexaweb.server.Application

```
public synchronized void setApplicationVariable(String id, Object ctrl)
 {
   if(ctrl instanceof java.awt.Component) {//need to add an ID field for
Component class
    // System.out.println("Putting CTRL="+id+", Object="+ctrl+" into the CTRL
table");
      ctrlTable.put(id,ctrl);
   }
 }

public Object getApplicationVariable(String id)
{ return ctrlTable.get(id); }

public void delApplicationVariable(String id)
{ ctrlTable.remove(id); }

private String getListenerKey(String ctrlId, String eid) {
  return ctrlId+eid;
}

public Vector getListeners(String ctrlId, String eventId) {
  String key=this.getListenerKey(ctrlId,eventId);
  Vector ls=(Vector)listenerTable.get(key);
  return ls;
}

/**
 *add a listener to be stored as application variable
 *@param ctrlId, the id of the source component of the event.
 *@param eventId, the id of the event type
 *@param listener, the listener object, who contains methods for event
processing
 */
public synchronized void addListener(String ctrlId, String eventId, Object
listener) {
  String key=this.getListenerKey(ctrlId,eventId);
  Vector ls=(Vector)listenerTable.get(key);
  if(ls==null) ls=new Vector();
  ls.addElement(listener);
  listenerTable.put(key,ls);
  //System.out.println("Add Listener to Application:
ctrlId="+ctrlId+",eventId="+eventId+",listener="+listener);
}

public synchronized void removeListener(String ctrlId, String eventId, Object
listener) {
  String key=this.getListenerKey(ctrlId,eventId);
  Vector ls=(Vector)listenerTable.get(key);
  if(ls==null) return;
  ls.removeElement(listener);
  listenerTable.put(key,ls);
}

public void start() throws Exception
  {ClassLoader cl=this.getClass().getClassLoader();
   System.out.println("entry="+appName+",appId="+appId+",class
Loader="+cl);
   Class entry;
   if(cl==null) entry=Class.forName(appName);
   else entry=Class.forName(appName,
   System.out.println("entry="+appName+","+entry),

   try {
     AppServiceThread thread=new AppServiceThread(this,entry,"main",null),
   System.out.println("thread="+thread);
   thread.run();
   }catch(Exception ex) {System.out.println("Application Start Exception,
"+ex);return;}

    /**
     *thread.start(),
     *We can not use thread.start() here because if you spawn off a new thread
to do the processing,
     * the original servlet service thread will just return and die. As a result, it will
close the
     *http response connection.
     */
   }
}
```

Class com.nexaweb.server.EventManager

```
package com.nexaweb.server;

import java.lang.*;
import java.lang.reflect.*;
import java.util.*;
import java.io.*;
import java.text.*;
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;

import javax.servlet.*;
import javax.servlet.http.*;

public class EventManager {

 public  EventManager() {return;}

public static int getEventID(String event) {
  if(event==null) return 0;
   /**
   *mouse events
   */
  if(event.equals("MouseDown")) return 10;
  else if(event.equals("MouseUp")) return 11;
  else if(event.equals("MouseOut")) return 12;
  else if(event.equals("MouseOver")) return 13;
  else if(event.equals("MouseDoubleClick")) return 14;
  else if(event.equals("MouseClick")) return 15;
  else if(event.equals("MouseDrag")) return 16;
  else if(event.equals("MouseDrop")) return 17;
  else if(event.equals("MouseMove")) return 18;
   /**
   *Action events
   */
  else if(event.equals("ActionEvent")) return 20;
  else if(event.equals("WindowEvent")) return 30;


  else return 20000;
  }

public static int stringToInt(String s) {
    Integer slot=(new Integer(s));
    int id=0;
    if(slot!=null) id=slot.intValue();
    return id;
}

public static void dispatchEvent(HttpServletRequest request,
                HttpServletResponse response,
                String appid,String ctrlid, String eid)
    throws IOException, ServletException
  {
    System.out.println("Entering dispatch event method...");

    Application app=AppManager.getApplication(appid);
    if(app==null) {
      System.out.println("Can not find application with ID="+appid);
    }

    Vector v=app.getListeners(ctrlid, eid);
    if(v==null||v.size()<1) return;

    Thread thread=java.lang.Thread.currentThread();
    String tname=thread.getName();
    System.out.println("Working ...Current thread name
="+tname+",eventID="+eid+",ctrlID="+ctrlid);
    HttpManager.put(tname,response);
    AppManager.addAppThread(tname, app.getAppId());

    Integer ints(new Integer(eid));
    int eventId=0;
    if(slot!=null) eventId=slot.intValue();
    System.out.println("Event ID="+eventId);

    if(eventId==getEventID("MouseDown")) {
      processMouseEvent(app, eventId,ctrlid,v,request,response);
    }
    else if(eventId==getEventID("MouseOver")) {
      processMouseEvent(app, eventId,ctrlid,v,request,response);
    }
    else if(eventId==getEventID("MouseOut")) {
      processMouseEvent(app, eventId,ctrlid,v,request,response);
    }
    else if(eventId==getEventID("MouseDown")) {
      processMouseEvent(app, eventId,ctrlid,v,request,response);
    }
    else if(eventId==getEventID("MouseUp")) {
      processMouseEvent(app, eventId,ctrlid,v,request,response);
    }
    else if(eventId==getEventID("MouseDoubleClick")) {
      processMouseEvent(app, eventId,ctrlid,v,request,response);
    }
    if(eventId==getEventID("MouseClick")) {
      processMouseEvent(app, eventId,ctrlid,v,request,response);
    }
    else if(eventId==getEventID("MouseDrag")) {
      processMouseEvent(app, eventId,ctrlid,v,request,response);
    }
    else if(eventId==getEventID("MouseDrop")) {
      processMouseEvent(app, eventId,ctrlid,v,request,response);
    }
    else if(eventId==getEventID("MouseDrag")) {
      processMouseEvent(app, eventId,ctrlid,v,request,response);
    }
    else if(eventId==getEventID("MouseMove")) {
      processMouseMotionEvent(app, eventId,ctrlid,v,request,response);
    }
    else if(eventId==getEventID("ActionEvent")) {
      System.out.println("Action Event: Calling processActionEvent");
      processActionEvent(app, eventId,ctrlid,v,request,response);
      System.out.println("Action Event: Finished processActionEvent");
    }
    else if(eventId==getEventID("WindowEvent")) {
      processWindowEvent(app, eventId,ctrlid,v,request,response);
    }

      HttpManager.remove(tname);
      AppManager.removeAppThread(tname);
      System.out.println("finished processing event. Thread="+tname);
  }
```

Class com.nexaweb.server.EventManager

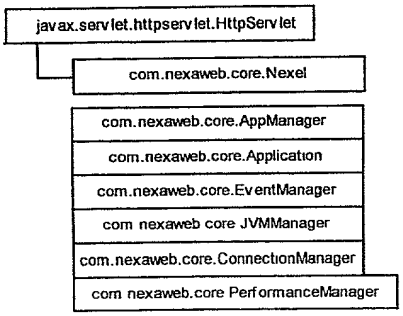
```
   public static void processMouseMotionEvent(Application app, int eid,String
cid,Vector listeners,HttpServletRequest request,
                HttpServletResponse response) {


  public static void processActionEvent(Application app, int eid,String
cid,Vector listeners,HttpServletRequest request,
                HttpServletResponse response) {

      System.out.println("Entering processing Action Event:
app="+app+",eid="+eid+",ctrl="+cid+",listeners="+listeners);
      if(listeners==null) return;
      if(app==null) return;

      for(Enumeration eu=listeners.elements();eu.hasMoreElements();)
        { Object o=eu.nextElement();
          if(eid==getEventID("ActionEvent")) {
              if(!(o instanceof ActionListener)) {
                  System.out.println("Listener is not of type 'ActionListener'
"+eid+",cid="+cid+",listener="+o);
                  return;
              }
              ActionListener al=(ActionListener)o;
              String cmd=request.getParameter("command");


              // System.out.println("processing action event:
command="+cmd);
              Object ctrl=app.getApplicationVariable(cid);

              //System.out.println("processing action event: CTRL="+ctrl);

              if(ctrl!=null) {
                if(ctrl instanceof AbstractButton)
                    cmd=((AbstractButton)ctrl).getActionCommand();
                else if(ctrl instanceof Button)
                    cmd=((Button)ctrl).getActionCommand();
              }

              String modifier=request.getParameter("modifier");
              int mask=0;
              if(modifier!=null) mask=stringToInt(modifier);
              // System.out.println("processing action event:
command="+cmd+",ctrl="+ctrl+",mask="+mask);
              ActionEvent event=new ActionEvent(ctrl,eid,cmd,mask);
              al.actionPerformed(event);
              System.out.println("processed action event...");
          }
        }
              }

   public static void processWindowEvent(Application app, int eid,String
cid,Vector listeners,HttpServletRequest request,
                HttpServletResponse response) {
      if(listeners==null) return;
      if(app==null) return;

      for(Enumeration eu=listeners.elements();eu.hasMoreElements();)
        { Object o=eu.nextElement();
          if(eid==getEventID("WindowEvent")) {
              if(!(o instanceof WindowListener)) {
                  System.out.println("Listener is not of type 'WindowListener'
"+eid+",cid="+cid+",listener="+o);
                  return;
              }
              WindowListener wl=(WindowListener)o;
              String modifier=request.getParameter("windoweventtype");
              int type=stringToInt(modifier);

              Object wo=app.getApplicationVariable(cid);
              if(!(wo instanceof Window)) {
                  System.out.println("Event source object is not of type
'java.awt.Window' "+eid+",cid="+cid+",listener="+o);
                  return;
              }
              Window win=(Window)wo;
              WindowEvent event=new WindowEvent(win,type);
              if(type==WindowEvent.WINDOW_ACTIVATED)
wl.windowActivated(event);
              else if(type==WindowEvent.WINDOW_CLOSED)
wl.windowClosed(event);
              else if(type==WindowEvent.WINDOW_CLOSING)
wl.windowClosing(event);
              else if(type==WindowEvent.WINDOW_DEACTIVATED)
wl.windowDeactivated(event);
              else if(type==WindowEvent.WINDOW_DEICONIFIED)
wl.windowDeiconified(event);
              else if(type==WindowEvent.WINDOW_ICONIFIED)
wl.windowIconified(event);
              else if(type==WindowEvent.WINDOW_OPENED)
wl.windowOpened(event);
              else {System.out.println("Window event type="+type+", is not
handled");}

              System.out.println("processed window event ...");
          }
        }
              }


}
```

Appendix 2: Nexel Server Class Diagram

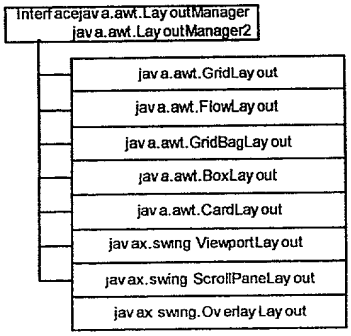
Packages :

- com.nexaweb.core: core classes for Nexel Server
- com.nexaweb.net: Nexel Network Engine for communicating with Nexel Client Kernel and other Nexel Servers
- Selected classes in package java.awt
- Seleced classes in package javax.swing
- com.nexaweb.plaf.ce: all the UI classes for Windows CE platform
- com.nexaweb.plaf.pc: all the UI classes for PC (Widnows desktop, Unix machine, Machintosh) platforms

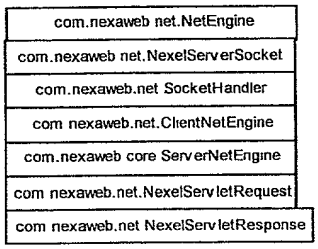
Packagecom.nexaweb.core(NexelCoreClasses)



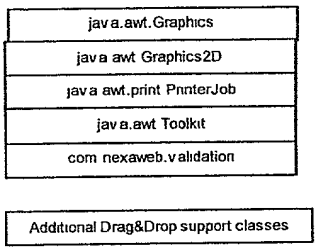
LayoutManagers

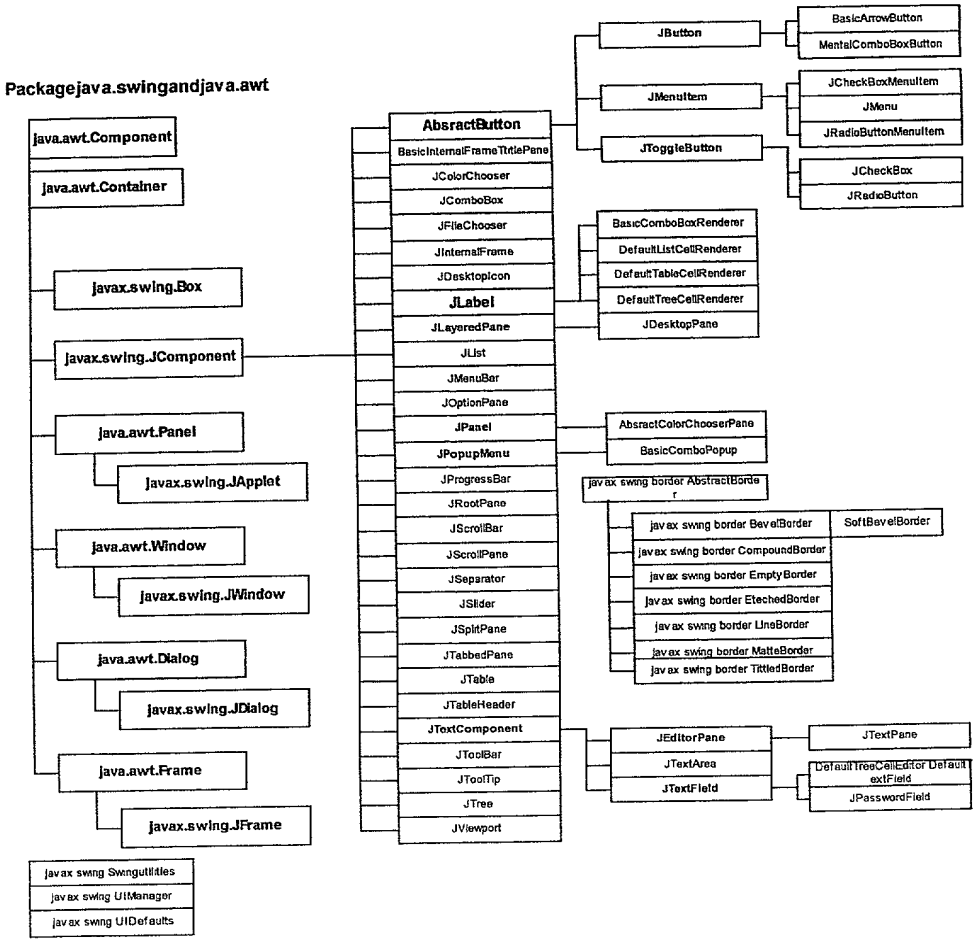


Packagecom.nexaweb.net

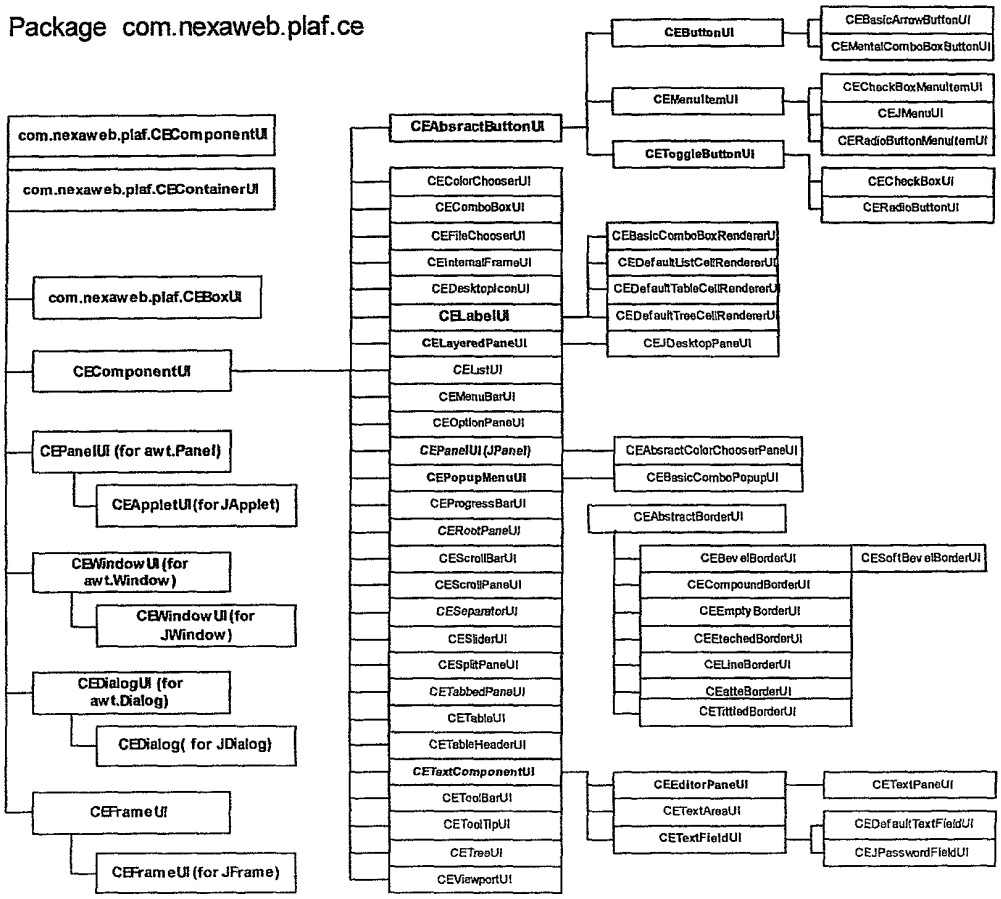


AdditionalClasses




Packagejava.swingandjava.awt
java.awt.Component
java.awt.Container
Javax.swing.Box
Javax.swing.JComponent
java.awt.Panel
Javax.swing.JApplet
java.awt.Window
Javax.swing.JWindow
java.awt.Dialog
Javax.swing.JDialog
java.awt.Frame
Javax.swing.JFrame
Javax swing Swingutilities
Javax swing UIManager
Javax swing UIDefaults
AbsractButton
BasicInternalFrameTitlePane
JColorChooser
JComboBox
JFileChooser
JInternalFrame
JDesktopIcon
JLabel
JLayeredPane
JList
JMenuBar
JOptionPane
JPanel
JPopupMenu
JProgressBar
JRootPane
JScrollBar
JScrollPane
JSeparator
JSlider
JSplitPane
JTabbedPane
JTable
JTableHeader
JTextComponent
JToolBar
JToolTip
JTree
JViewport
JButton
BasicArrowButton
MentalComboBoxButton
JMenuItem
JCheckBoxMenuItem
JMenu
JRadioButtonMenuItem
JToggleButton
JCheckBox
JRadioButton
BasicComboBoxRenderer
DefaultListCellRenderer
DefaultTableCellRenderer
DefaultTreeCellRenderer
JDesktopPane
AbsractColorChooserPane
BasicComboPopup
javax swing border AbstractBorde r
javax swing border BevelBorder
SoftBevelBorder
javax swing border CompoundBorder
javax swing border EmptyBorder
Javax swing border EtechedBorder
javax swing border LIneBorder
Javax swing border MatteBorder
Javax swing border TittledBorder
JEditorPane
JTextArea
JTextField
JTextPane
DefaultTreeCellEditor Default extField
JPasswordField

Package com.nexaweb.plaf.ce



Package com.nexaweb.plaf.pc